VŠB – Technical University of Ostrava
Faculty of Electrical Engineering and Computer Science
Department of Computer Science

Malware and the Possibilities of its Evolution

Malware a možnosti jeho evoluce

2019 Jakub Šustr

VŠB - Technical University of Ostrava
Faculty of Electrical Engineering and Computer Science
Department of Computer Science

Diploma Thesis Assignment

Student: **Bc. Jakub Šustr**

Study Programme: N2647 Information and Communication Technology

Study Branch: 1801T064 Information and Communication Security

Title: Malware and the Possibilities of its Evolution
Malware a možnosti jeho evoluce

The thesis language: English

Description:

The thesis is focused on the use of bioin spirits algorithms in the field of testing of evolution and mutation of computer malware. The aim and purpose of the work is to experiment with an evolution of computer malware with a focus on viruses in an isolated virtual environment. The work will be a continuation of the existing work and will be linked to it. The work is expected to experiment with metamorphic viruses and their evolutionary evolution with the help of a superior evolutionary process. Evolutionary algorithms will use PSO, SOMA, DE and GA.
Evolutionary algorithms are a neophyte for identifying algorithms that use Darwin's theory of evolution and Mendel's inheritance theory to simulate natural processes to solve complex optimization problems. In this case, virus synthesis from predefined building blocks will be performed. The output of the work will be: a functional program enabling the evolution of the virus with evolutionary algorithm selection, data obtained from experiments and their statistical and graphic processing.

The expected structure of the work is:
1. Getting to know the issue.
2. Selecting a suitable programming environment and designing experiments.
3. Selection of suitable algorithms in the field of evolutionary techniques and creation of necessary building blocks of metamorphic viruses.
4. Program implementation of these algorithms.
5. Visualization of all implemented algorithms.
6. Creating a user manual.

References:

[1] Merhaut F., Zelinka I., Úvod do počítačové bezpečnosti, Fakulta aplikované informatiky, UTB ve Zlíně, Zlín, 2009
[2] Peter Szor, Počítačové viry - analýza útoku a obrana, Zoner Press
[3] Zelinka I., Oplatková Z., Šeda M., Ošmera P., Včelař F., Evolutionary techniques – principles and applications, BEN, Prague, 2008, 598 p.
[4] Zelinka Ivan , Artificial Intelligence, chapter 6 „Differential evolution“, Academia, Prague, 2004, 33 p.
[5] Koza J.R.,Bennet F.H., Andre D., Keane M. 1999, Genetic Programming III, Morgan Kaufnamm pub.,ISBN 1-55860-543-6, 1999
[6] Kvasnička V., Pospíchal J., Tiňo P. 2000, Evolučné algoritmy, STU Bratislava, ISBN 85-246-2000, 2000

Extent and terms of a thesis are specified in directions for its elaboration that are opened to the public on the web sites of the faculty.

Supervisor: **prof. Ing. Ivan Zelinka, Ph.D.**

Date of issue: 01.09.2018

Date of submission: 30.04.2019

doc. Ing. Jan Platoš, Ph.D.
Head of Department

prof. Ing. Pavel Brandštetter, CSc.
Dean

I hereby declare that this master's thesis was written by myself. I have quoted all the references I have drawn upon.

Ostrava, April 23, 2019 .....

I hereby agree to the publishing of the master's thesis as per s. 26, ss. 9 of the Study and Examination Regulations for Master's Degree Programmes at VŠB – Technical University of Ostrava.

Ostrava, April 23, 2019 .....

I would like to thank prof. Ing. Ivan Zelinka, Ph. D. for comments and suggestions and overall work leading.

Abstrakt

Tato práce je zaměřena na použití bio-insipirovaných algoritmů v oblasti evoluce malwaru. V úvodu práce je seznámení s problematikou malwaru a problematikou evolučních algoritmů. Dále je zde popis vytvořeného počítačového viru, využívajícího evoluční techniky a principy modulárního programování. V poslední části této práce je popsán proběhlý experiment, a také jeho výsledky.

Klíčová slova: malware, virus, evoluční algoritmy, modulární programování, diplomová práce

Abstract

The aim of this work is focused on using bio-inspired algorithms in field of malware evolution. In the beginning of this work is introduction with malware and evolutionary algorithms problematic. Next part of this work contains description of created computer virus, which uses evolutionary techniques and modular programming principles. In last part of this work is described experiment performed in this work and its results.

Key Words: malware, virus, evolution algorithms, modular programming, master thesis

Contents

List of symbols and abbreviations

DVD	–	Digital Versatile Disc
PE	–	Portable Executable
HW	–	Hardware
EA	–	Evolutionary Algorithm
PSO	–	Particle Swarm Optimization
SOMA	–	Self-Organizing Migrating Algorithm
DE	–	Differential Evolution
GA	–	Genetic Algorithm
OS	–	Operation System
I/O	–	input/output
TCP	–	Transmission Control Protocol
DoS	–	Deny of Service
DDoS	–	Distributed Deny of Service
API	–	Application Program Interface
CPU	–	Central Processing Unit
HDD	–	Hard Disk Drive
PE	–	Portable Executable
FPU	–	Floating Point Unit
SEH	–	Structured Exception Handling

List of Figures

List of Tables

Listings

1 Introduction

1.1 Motivation

Due to the ever-expanding number of computer systems, there is, and there will be more and more malware targeting these systems. Evolvable malware looks very interesting and terrifying at the same time. Not only that evolvable malware will be heavy challenge for anti-virus products because anti-viruses in these days still using as a core part of recognizing malware pattern matching and evolvable malware will change its signature (whole or particularly) in every generation. But it will be challenging for heuristic part of anti-viruses too because malware will be able to change its behaviour too.

1.2 Looking on evolvable malware

We can look at evolvable malware in two different views.

First view is that we can look on it as the type of malware defense against anti-viruses. In history we see that there is some kind of malware defense evolution. We can divide this defense to five categories and as next (six) category in this camouflage evolution we can consider evolvable malware. These five categories are shown in Figure 1.

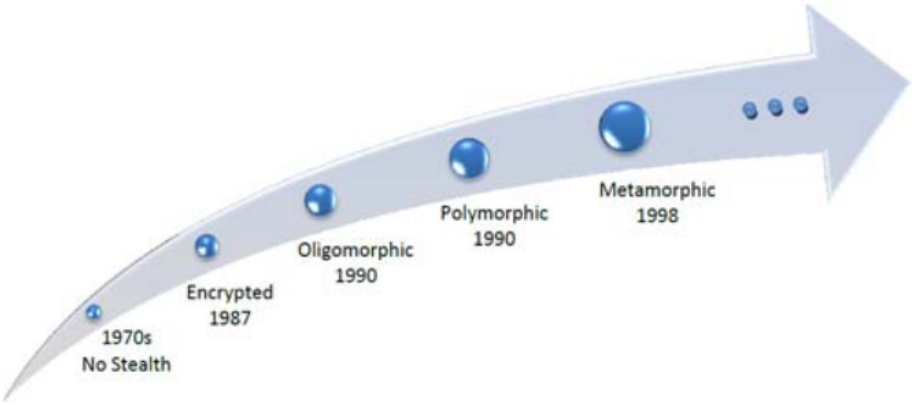

Figure 1: Malware camouflage evolution

Source: [4]

The second view is that how we can look on behaviour of malware. Because evolvable malware can adapt to environment it can be interesting to see malware which can "decide" that for better accomplishment of the given task it will be better to change its behaviour. For example we can have malware which task will be "make money". That malware can in some environment mine bitcoins (e.g. machine with good HW) and the same malware in the other environment can act as ransomware (e.g. machine with many documents on disk).

2 State of the art

2.1 Brief history

As first concepts on computer viruses can be considered John von Neumann's self-reproducing mathematical automata in the 1940s and in 1951 Neumann shows methods how to create this automata. Malicious software (malware) started to be popular with the expansion of computer networks. Before that, malware existed too but was not so widely spread because there was not such opportunity as is with computer networks.

Nowadays malicious software primary exploits Microsoft Windows. In the following lines will be described a brief history of evolving malicious software over the last few decades.

2.1.1 1960s

In 1962 group of engineers (V. Vyssotsky, G. McIlroy, R. Morris) created game "Darwin" which goal was to battle about computer using programs and multiply against opponents programs. Players need to eliminate opponents programs to win the game.

2.1.2 1970s

Early in 1970s was detected virus "Creeper" on US military network called "ARPANET". Creeper was written on Tenex OS and was capable to copy itself through modem to remote location. Infected system displayed message "I'M THE CREEPER : CATCH ME IF YOU CAN.".

Shortly after that was created virus "Reaper" which goal was to delete "Creeper" virus. It is unclear if this virus was written by the same person as "Creeper" or it was work of someone else.

In 1974 virus "Rabbit" was created. It gets its name by the payload it does. Rabbits only payload was to multiply and spread, but it was doing it so fast that performance issues occurred.

In 1975 was created "Pervading Animal" game in which player think of animal and game tries to identify it. When the game was unable to guess the animal its update itself and enter new questions. This updated version overwrites old one and copies itself to other directories. It is unclear if it was a malicious intention of the author or if it was badly programmed program.

2.1.3 1980s

In 1981 virus "Elk Cloner" was created. This virus was infecting Apple II OS. The virus started when the computer was booted from an infected floppy disk. When was inserted non-infected floppy disk the virus copies itself on it. As a payload virus uses rotating images, blinking text and joke messages.

In 1983 was term "virus" used in connection with self-replication computer program by L. Eidelmen and showed at seminar virus-like program. In 1984 L. Eidelmen defined computer virus as program which is able infect other programs and replicate itself to them.

In 1986 was created virus "Brain". Virus spread itself on IBM compatible systems and in a few month world outbreak erupted. Brain uses boot sector for infection. Authors of virus (B.F. Alvi and his brother Amjad) worked in sales for a software company in Pakistan. The virus contained theirs names and phone number. Virus "Brain" can be marked as the first "stealth" virus - in case that someone wants to show boot sector, the virus shows original uninfected boot sector.

In 1986 (in the same year as Brain) was released virus named "Virdem" by R. Burger (german programmer). His virus was able to copy itself by adding its code to code executable DOS files in COM format.

In 1987 several viruses were created.

- Vienna - its payload shows Christmas tree.
- Lehigh - installing itself to memory from infected disk with COMMAND.COM file.
- Suriv virus family
- Boot sector viruses: Yale, Stoned, Ping Pong
- Cascade - first self-encrypted virus. The virus consists of body and encryption routine. Virus payload was that text of screen cascade down and form a heap on the bottom.

In 1988 virus Jerusalem (Suriv-3) was released. This virus was memory resistant and inflicted every executable file. This virus also had payload which triggers on 13th Friday except in the year 1987. When a condition is met virus delete every file that was executed.

In the same year (1988) was released computer worm Morris. This worm exploited known vulnerabilities on Unix and harvest passwords. The virus sends an unlimited number of copies of itself and completely overloaded the networks. The overall losses were estimated to $96 million dollars[10].

In 1989 was released WANK worm. Worm spread via DECnet protocol. Worm payload was message "WORMS AGAINST NUCLEAR KILLERS" and "Your System Has Been Officially WANKed.". This worm also changed system password and sent it to user "GEMPAK" on SPAN network.

In the same year was released "AIDS" trojan. 20000 disc were sent as Aids Information Diskette. After infected disk was loaded trojan installs itself on the system and start to conceal files and directories and modify system files. After 90 loads trojan leaves only one file visible which contains information to pay money to the bank account.

2.1.4 1990s

In 1990 the first polymorphic virus was released. Virus family Chameleon was encrypted, but on top of that, it changes virus code with every infection.

In July 1990 DiskKiller virus was infected on a free floppy disc which was in PC Today computer magazine. More than 50000 were sold.

Frodo virus appears in 1990 too it uses stealth techniques to hide. Its payload was message "FRODO LIVES!" with pattern moving around.

Whale virus released in 1990, it was 9,216 bytes large virus which uses multiple encryption and other advanced stealth techniques to hide itself in system.

In 1991 was created virus "Tequila" it was polymorphic boot infector, causing an epidemic. This virus was written by a Swiss programmer for research purposes but was stolen by acquaintance and spread.

In 1992 was created virus "Peach" which was one of the first retroviruses. It deletes the database of antivirus, so antivirus "thinks" that it is launched first time and recreate the whole database slowly again. This way Peach can slowly inflict system without being noticed.

In the same year (1992) was created the first Windows virus "Win.Vir_1_4".

In 1994 were created viruses "SMEG.Pathogen" and "SMEG.Queeg". It were highly polymorphic viruses which causes data loses (viruses overwrite part of the disk).

In 1996 was created the first virus for Windows 95 named "Boza". It affects Windows executable files (EXE). Boza infects 3 EXE files in the directory where is launched and after that launch original file. Boza payload is windows with a message containing information about virus writers.

In July 1996 was founded first Microsoft excel virus which uses macros written in Visual Basic programming language.

In February 1997 the first virus for Linux appeared named "Bliss".

In March 1997 appeared virus "ShareFun" which was the first kind of virus which uses spreading via email.

In April 1997 virus "Homer" appeared it was the first network worm which uses FTP to propagate itself through the network.

In January 1999 appeared worm "Happy99" which caused global epidemic. It uses MS Outlook to spread itself. Worm shows animated firework as its payload and modify registry to start after system boot.

In December 1999 first self-rejuvenation worm appeared. It was worm Babylonia. Worm was capable of download newer version of its modules from remote location from Japan. It connects to server to Japan every minute to check if there is newer version of its modules and if there were it download and install them.

2.1.5 2000s

In May 2000 appeared virus LoveLetter. The virus was written in visual basic and spread by email as "LOVE-LETTER-FOR-YOU.TXT.vbs" attachment. The virus then set homepage of Internet Explorer to one of the predefined and from there downloaded another its part which placed itself in Windows system directory and harvest sensitive information from the computer and sent it to specified email address.

In September 2000 appeared virus named "Stream" which was able to work with ADS (alternate data stream) of NTFS file system.

In November 2000 appeared virus "Hybris" which was enhanced virus Babylonia. The main enhancement was the use of websites and list of servers and using 128-bit RSA key for identifying modules.

In 2001 appears "fileless" worms. These worms exist only in RAM, and it doesn't need files to replicate and spread itself.

In the same year appeared worm Code Red. This worm exploits a vulnerability in Microsoft IIS and causes DDoS attacks on several fixed IP addresses.

In 2002 Linux worm Slapper appeared and infected thousand Linux systems. The worm used a vulnerability in OpenSSL through connection to Apache.

On January 25th, 2003, worm Slammer appeared. This worm caused increased traffic over the network by 40 % - 80 % and inflicted hundreds of thousand computers within a few minutes. Slammer exploited buffer overflow in Microsoft SQL server (patch for this vulnerability was available half of the year at the time of worm started to spread).

In August 2003 Blaster (Lovesan) worm appeared. The worm used buffer overflow to spread itself and caused DDoS attack on website windowsupdate.com which was in fact only redirecting page to windowsupdate.microsoft.com.

In January 2004 email worm Bagle appeared. Bagle was used to create Trojan proxy servers on victim machines.

In April 2004 worm Sasser appeared. Sasser exploited loophole in LSASS (Local Security Authority Subsystem Service) in Microsoft Windows. Sasser paralyzed many businesses including banks, airlines, hospitals etc.

In October 2005 was discovered rootkit which uses Sony company on their discs to prevent illegal copying. When disc was inserted to pc it installs software which modify operating system to make copying discs harder. This rootkit created vulnerabilities which were immediately used by virus writers.

In November 2008 worm Conficker appeared. Conficker used many advanced malware techniques. It uses RPC (remote procedure call) exploit which contains exploit to buffer overflow which then downloaded worm as .jpg file. After worm was on victim systems it checks its public IP address by using on of the several website for this purpose. After that Conficker create HTTP

server on random port, infect another victim and let the victim download Conficker from newly created HTTP server.

2.1.6 2010s

In June 2010 Stuxnet worm was detected. Stuxnet targeted only specific devices (Siemens centrifuges) causing them to self-destruct. Stuxnet used 4 "zero-day" vulnerabilities to its operation and had switch off date which was set on June 24, 2012. Stuxnet was able infect not only using network, but can infect systems using removable devices too.

In September 2013 ransomware "Cryptolocker" appeared. It is considered as one of the first ransomware. It was spread by zip attachments in email. In zip archive was EXE file which was named *.pdf.exe and has pdf icon. After launch Cryptolocker start encrypts victim files and demanded ransom to be paid to unlock files. It was one of the ransomware that actually gave to user decryption key if ransom was paid.

In 2015 Bashlite malware appeared exploiting "Shellshock" software bug and causing several DDoS attacks.

In 2016 ransomware Locky appeared. It was contained as macro of Microsoft Word document pretending to be an invoice.

In September 2016 botnet Mirai showed its capability. Mira caused several massive DDoS attacks (to 1Tbit/s). Mirai abused the fact that on many IoT devices were never changed login credentials. Mirai load itself to memory and after loaded it deletes itself from disk so theoretically after the reboot was device free of virus, but in time of attacks, device was again infected in a few minutes after reboot if credentials remain unchanged.

In 2017 another ransomware appeared. Ransomware WannaCry and Petya used leaked NSA hacking tools to exploits to targeted systems. Both mentioned used EternalBlue exploit which was exploiting a vulnerability in Microsoft SMB protocol for remote code execution.

2.2 Current state of the art

2.2.1 Metamorphic malware

Metamorphic viruses are viruses that are one of the most advanced and complicated viruses and may be labeled as the best (in term of advanced techniques and difficulty to detect them) viruses yet. Theses viruses using advance techniques to be able to change its structure. This includes abilities to edit and rewrite its code, but remain its functionality. Due to this fact it is very hard to detect a metamorphic virus by its signature because the signature is changing over time.

Changing structure is performed by metamorphic operations such as register usage exchange, code permutation, code expansion, code shrinking and insertion of garbage code.

Register usage exchange is a technique where are in different iterations of virus used different registers, so functionality remains the same, but the code is different due to usage of different registers.

Code permutation is a technique where virus code is divided into frames which can be permuted. The virus then ensures that these frames are executed in the correct order.

Code expansion is a technique where the virus will expand its logic to more instructions, but ensure that logic remains the same.

Code shrinking is a technique which is similar to expansion, but instead of expanding to more instructions it is shrunk to less instructions. This technique is quite hard because in fact to get less instructions with the same functionality it is needed to optimize code.

Inserting garbage code is a technique where it is inserted garbage instructions which do nothing in term of functionality, but the code is changed.

These techniques listed above provide metamorphic malware with certain resilience against signature-based scanning of anti-malware programs.

2.2.2 Evolving malware

Current evolvable malware is in the its beginning. There were not many experiments in this field at least what author know. But that few which were done we could actually consider as current state of the art.

We may see that sometimes is metamorphic malware labeled as evolvable malware. Author opinion is that we cannot label metamorphic malware as evolvable malware. Metamorphic malware is malware which changes its structure in every iteration to camouflage itself, but evolutionary algorithms are not necessarily needed for this purpose, and it can be achieved without them.

In one of the experiment done by S. Noreen, S. Murtaza, M. Z. Shafiq, M. Farooq. They wrote evolutionary framework for evolving malware. In their experiment they were trying evolve virus based on virus family "Bagle".[2]

Another experiment was done by A. Cani, M. Gaudesi, E. Sanchez. Where they were working with the code of the virus "Timid". In their experiment they used genetic algorithm to find best place to inject virus code into PE application.[3]

In the light of these few experiments, we can say that current state of the art of evolvable malware is ability to use evolution in its part or to use evolution as a tool to change the way how is virus working but still remaining its origin functionality.

3 Theoretical part

Theoretical part of this work is divided to 4 sections. These section are: Evolutionary algorithms, Types of malware, Virus defense techniques and past experiments.

3.1 Evolutionary algorithms

In this thesis are used evolutionary algorithms for generating malware blocks and simulating evolution. There are used 4 algorithms: PSO, SOMA, DE and GA. We will describe each of them more in the following paragraphs.

3.1.1 PSO

Particle swarm optimization is optimization technique which works with population like genetic algorithms. This technique was founded by Russel Eberhart and James Kennedy in 1995 and its inspired by social behaviour of animal communities like bird and fish swarm.

Unlike some other evolutionary algorithms the particle swarm does not use selection. The problem solving is done by interactions between all population members which results in improving solution over time.

3.1.1.1 Principle

As it was said particle swarm is simulating behaviour of animal swarms like bird. We can easily imagine its function as swarm of birds which are looking for the highest place to sit. Each bird in the swarm is represented as particle in PSO. Birds don't know where the highest place is, but they know where is the highest bird from the swarm. So the others birds (particles in PSO) will follow its direction.

The same principle is used in PSO algorithm. All particles have their coordinates in searched space, their velocity and they all know their last best position. The best position is determined using the fitness function.

The first population in PSO is initialized randomly with randomly chosen parameters (velocity vector, coordinates) and it is computed by fitness function their value. The best particle is stored in shared memory so each particle can know where is the best solution yet. This value in population is called "gBest" (g-global).

After each iteration, all particles will compare its actual fitness value with its previous personal best value, and if the actual fitness value is better, it will be replaced by new best value. And then all particles will recalculate the velocity vector and position.

When all particles known their personal best (pBest) and global best (gBest) they will recalculate their velocity vector and position using Equation 1 and Equation 2.

$$v_d(t+1) = v_d(t) + c_1 * rand * (pBest_{i,d} - x_{i,d}(t)) + c_2 * rand * (gBest_d - x_{i,d}(t)) \quad (1)$$

$$x_{i,d}(t+1) = x_{i,d}(t) + v_d(t+1) \quad (2)$$

Where:
$v_d(t+1)$ - velocity of particle in the next step
$v_d(t)$ - velocity of particle in the current step
$x_{i,d}(t+1)$ - position of particle in the next step
$x_{i,d}(t)$ - position of particle in the current step
$pBest_{i,d}$ - the current best position of the current particle
$gBest_d$ - the current best position in the population
$rand$ - random number in interval $(0, 1)$
$c1, c2$ - learning factors (usually it is 2)

We can say that particles can take 3 ways (shown in Figure 2):

- Individual: particles continue on their way
- Conservative: particles return to its current best position
- Adaptive: particle follow particle which found the current best global position

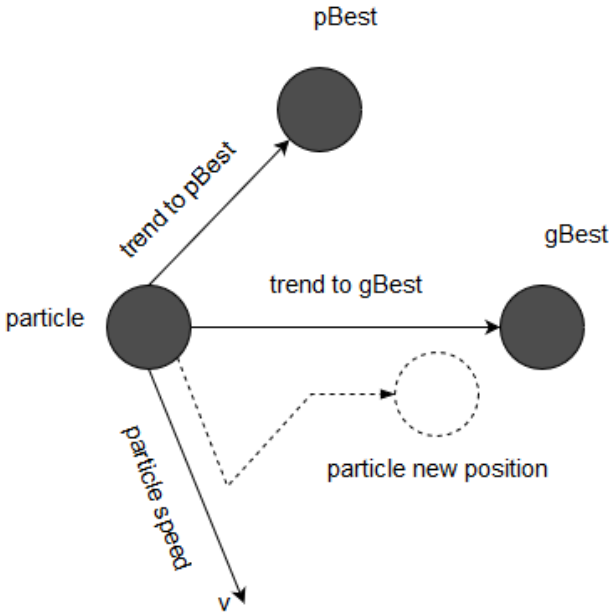

Figure 2: Particle moves influenced by its trends

3.1.1.2 Algorithm

Principle of PSO can be demonstrated in pseudo-code (Listing 1). In this example we are not checking if particles are in searched space nor if particle velocity exceeds max velocity.

```
{
   Input:
   iteration: number of iterations
   particles: number of particles p_i
   gBest: best founded solution in population
   pBest: best founded solution of current particle
}

{ random initialization }
for j =< particles do
  begin
    p_j = rand_j
  end

for i < iteration do
  begin
    for j =< particles do
      begin
      { calculate speed of particle (equation 1) }
      { calculate positon of particle (equation 2) }
      fitness = fitness_function(p_i)
      if fitness < pBest then
        begin
          pBest = fitness
        end
      if pBest < gBest then
        begin
          gBest = pBest
        end
      end
  end
```

Listing 1: PSO pseudocode

3.1.1.3 Adjustable parameters

- **Dimension** – number of inputs to fitness function. Dimension of solving problem.
- **Range** - boundaries of searched space.
- **Particle count** - number of particles (population size). Recommended value is 10× dimension.
- V_{max} - maximal velocity of particles. Recommended value is 1/20 of range.
- **Learning factors** c_1**,** c_2 - these learning factors influences trend of particles moves, but this influence is only partial. Factor c_1 is preference to returns back to "pBest". Factor Factor c_2 is preference to move to "gBest". Usually are both factors set to value 2.
- **Inertia weight (w)** - small value of inertia support searching local extremes and big value of inertia support searching of global extremes. Inertia weight (Equation 4) is used in calculation of next step velocity (Equation 3).

$$v_d(t+1) = w * v_d(t) + c_1 * rand * (pBest_{i,d} - x_{i,d}(t)) + c_2 * rand * (gBest_d - x_{i,d}(t)) \quad (3)$$

$$w = w_{start} - \frac{(w_{start} - w_{end}) * Iteration}{Migration} \quad (4)$$

Where:
$Iteration$ - number of round in migration rounds
$Migration$ - total amount of migration rounds
w_{start} - starting inertia weight (usually 0.9)
w_{end} - ending inertia weight (usually 0.4)

- **Constriction factor** - some PSO version using this instead of inertia. This factor has similar effects to inertia. Usual value is 0.729.

3.1.1.4 PSO disadvantages

- Often happens that too much initial velocity is generated and thus particles move away from best solution quickly. (Solved by limitation of max velocity).
- On problems with many optimums algorithm has tendency to premature convergence.
- Too many adjustable parameters.

3.1.1.5 Other versions of PSO

- **Neighborhood** - version of PSO which aiming to limit premature convergence.
- **Speciation** - version of PSO which introduces particles "species".
- **Niching** - version of PSO which creating sub-swarms to searching on local extremes while rest of swarm continues in searching further off.
- **INPSO** - Independent Neighborhoods Particle Swarm Optimization - this version using more neighborhoods which are independent of each other.
- **Hybrid** - combination of **INSPO** and **GPEA** (Geometrical Place Evolutionary Algorithm).
- **Dynamic Neighborhood** - version in which are particles in independent neighborhoods, but particles can change its neighborhood.

3.1.2 SOMA

SOMA is an algorithm which is based on vector operations like in scatter search, differential evolution or particle swarm optimization. So we can classify this algorithm as a swarm algorithm.

Due to the philosophy of algorithm, the "classic" generation is there called migration, but in fact it has the same meaning as the generation.

In biology analogy, we can say that its harem like reproduction system in swarm rather than a classic way of choosing parents from population for crossover.

3.1.2.1 Principle

SOMA was founded on the principle, which we can see in nature. It is inspired by competitive-cooperation behaviour of intelligent individuals solving a specific problem. We can see this type of behaviour in nature in various forms, for example, ants, bees, predators hunting in packs etc.

In the competitive phase is each individual trying to find the best solution to the given problem (e.g. find food).

In cooperation phase are individuals telling each other their quality (e.g. how much food they found) and they try to adapt their behaviour to be better in the next migration.

These two phases are continually repeating until end condition will occur. By these 2 simple phases we get **simple** model of this algorithm.

Before we start the algorithm we need to setup initial controlling parameters. These parameters are:

- **PathLength** - $(1; 5]$ - this parameter determine how far is active individual stop from leader individual. If PathLength is equal to 1, active individual will stop exactly on

location of leader individual. If is this parameter greater than 1 active individual will stop behind leader and vice versa. Recommended value is 3.

- **Step** - $[0.11; PathLength]$ - this parameters determine how fine or rough is sampling. The less the value is the better is chance to find global extreme, but the more resources (cost function calling) it costs. It is recommended to set this so it isn't integer multiple of distance (PathLength) between leader and active individual or our active individual can end exactly at leader location and diversibility of population will be lesser.
- **PRT** - $[0, 1]$ - PRT mean perturbation. This controlling parameter affects creating perturbation vector (PRTVector), which influences direction of which individual is heading when going to leader. If set to 1 active individual will take direct way (route) to leader. Recommended value is 0.1.
- **D** - Dimension of the given problem (number of arguments in cost function).
- **PopSize** - $[10; userdefined]$ - number of individuals in population. Recommended value is between $(0.2 \times D)$ to $(0.5 \times D)$ if D is high enough.
- **Migration** - $[10; userdefined]$ - number of migration rounds to proceed. Basically this parameter is one of the end conditions.
- **MinDiv** - $[userdefined; userdefined]$ - minimal diversity. It is second end condition parameter. This parameter specifies how big is difference between best and worst individual solution. If difference is smaller than this parameter, the algorithm will be ended. If we specified MinDiv as value lesser or equal to zero this condition will never occur and thus algorithm will end when all migrations end.

Process of mutation of this algorithm is similar to the other evolutionary algorithms with the exception that mutation is renamed to perturbation (again due to algorithm philosophy, functionality is the same). As we can see in pseudo-code (Listing 2).

```
for i:= 1 to N do
begin
   if rnd_j < PRT then
      PRTVector_j = 1
   else
      PRTVector_j = 0
end
```

Listing 2: SOMA perturbation vector pseudocode

Crossover is performed unlike the others algorithms, where it is performed by the crossover of 2 parents, by choosing the best offspring from steps taken by an active individual by its way to the leader. With this philosophy of crossover, we can define the equation of directional vector (Equation 5).

$$\vec{r} = \vec{r_0} + \vec{m}\, tPRT\vec{V}ector\,, t \in [0, Step, PathLength] \tag{5}$$

3.1.2.2 Algorithm

Following pseudo-code (Listing 3) is describing the basic version of SOMA - All to one.

```
{
   Input:
   x: first random generated population
   f_cost: cost function returning fitness of actual solution
}

for i < Migrations do
  begin
    {choose best individual - Leader}
  for j =< PopSize do
    begin
      {take j individual}
      {get fitness of new position}
      {choose best fitness individual and put it in new population}
    end
  if MinDiv < absolute_value(best_individual - worst_individual) then
    begin
      Exit()
    end
  end
```

Listing 3: SOMA All to one pseudocode

3.1.2.3 SOMA disadvantages

Amongst the biggest disadvantages belongs fact, that SOMA has too many adjustable parameters. There were even attempts to use evolutionary algorithms to tune SOMA parameters.

3.1.2.4 Other versions of SOMA

Except for basic version "**All to one**" where all individuals heading to leader. There are few other versions:

- **All to All** - In this version of SOMA doesn't exist Leader. All individuals are heading gradually to all others. Migrations ends after last individual end its migration. After that all individuals take new best positions which they found. This version is taking more resources, but have better chance to find global extreme due to greater sampling of hyper-surface.
- **All to All Adaptive** - This version is similar with version **All to All**, but the difference is that individual take new best position immediately after its migration ends - not after all individuals migration ends.
- **All to One Rand** - In this version all individuals heading to Leader, but Leader isn't the best individual on hyper-surface, but it is randomly chosen for each individual separately.
- **Clusters** - This version is applicable on all previous versions. It is basically only distribution individuals to sub-populations. In each sub-population is running independent SOMA. With regard to individuals are moving it regarding to clusters can disintegrate to more clusters and unite with other clusters.

3.1.3 Differential Evolution

Differential evolution is a type of algorithm founded in 1995 by K. Price and R. Storm. It has similar schema as genetic algorithms, but instead of taking two parents for creating offspring it takes four parents. In DE is performed mutation first and then is performed crossover, this is different from others evolutionary algorithms.

3.1.3.1 Principle

Population in DE is similar to the other evolutionary algorithms where the first population is generated randomly inside of specified boundary based on the given specimen.

In DE mutation using four parents instead of two. For every individual are randomly chosen three other individuals (not same as the first one). Using this three individuals (r_1, r_2, r_3) is created noisy vector (v), which is combination of mutation these 3 individuals. We can define noisy vector by Equation 6:

$$v_j = x^G_{r3,j} + F(x^G_{r1,j} - x^G_{r2,j}) \tag{6}$$

Where F is mutation constant and G is number of generation.

Crossover in DE is using 4. parent (r_4) and noisy vector (v) to create new individual (trial). This trial individual is created using crossover threshold (CR) so that there are taken parameters in r_4 and noisy vector (the same ones) in cycle and for each pair of parameters is generated random number. When this random number is lesser than CR then trial relevant parameter is set to value from noisy vector and vice versa. The condition for CR can be swapped without impact to performance. This new created individual is compared with r_4 and better one is taken to new population.

The whole principle of DE we can summarize to the following steps:

- **Parameter settings** - set parameters necessary for DE run
- **Population** - making starting population (usually randomly generated inside boundaries and inspired by specimen)
- **Generation cycle** - in each generation is cyclically taken every individual in the population, and it is performed evolution on it
- **Actual evolution cycle** - on each individual in previous step is performed evolution (explained earlier)
- **End condition test** - in basic DE there is only one end condition, and it is number of iteration, so basically, its only test if all generations were done
- **Evaluation** - if we want to show process of DE we can take every best individual in the generation and then show it in the graph

These steps 3-5 or 3-6 (if the process of DE is needed to be shown), are repeated until all generations are executed.

3.1.3.2 Algorithm

Following pseudo-code (Listing 4) is describing basic version of DE Rand1Bin.

```
{
   Input:
   x: first random generated population
   f_cost: cost function returning fitness of actual solution
}

for i < Generations do
  begin
  for j =< NP do
    begin
      {take j individual}
      {randomly take 3 individuals from population (not same as j)}
      {do mutation}
      {do crossover}
      {get fitness of trial}
      {compare trial with r4 and put better one to new population}
    end
    {replace old population by new one}
```

```
end
```

Listing 4: DE Rand1Bin

3.1.3.3 Adjustable parameters

In DE there are few parameters on which matters process of evolution. These parameters are:

- **CR** - $[0, 1]$ - Crossover threshold. In case that CR is set to 0 mutation is not performed and trial individual is created by copy of r_4 parent. Otherwise if CR is set to 1 trial individual is created only by first three parents (r_1, r_2, r_3) and DE will perform more like random search than evolutionary algorithm. Recommended value $0.8 - 0.9$. If is problem separable $CR << 1$, otherwise $CR \approx 1$.
- **D** - Problem dimension - number of arguments in cost function.
- **NP** - $[10D, 100D]$ - Size of the population. If it is set lesser than 4 DE will not work. Recommended value is $10D$.
- **F** - $[0, 2]$ - Mutation constant. Recommended value $0.3 - 0.9$.
- **Generations** - Number of generations in DE.

3.1.3.4 DE disadvantages

One of the primary disadvantages of DE is stagnation. Stagnation is in evolutionary algorithms known as premature convergence. Premature convergence occurs when:

- population is in local extreme
- population lost diversity
- optimization process occurs slowly or not at all

However in some cases of DE, there is stagnation even if all these previous conditions are negative.

3.1.3.5 Other versions of DE

There are several other versions of DE. We will show how is computed the noisy vector in some of them. Noisy vector in these variants are computed the same in bin and exp version, but there is difference how to fill trial in these versions.

- **DE/best/1/exp, DE/best/1/bin** (Equation **??**)

$$v = x^G_{best,j} + F \cdot (x^G_{r2,j} - x^G_{r3,j}) \tag{7}$$

- **DE/rand/1/exp, DE/rand/1/bin** (Equation 8)

$$v = x^G_{r1,j} + F \cdot (x^G_{r2,j} - x^G_{r3,j}) \tag{8}$$

- **DE/rand-to-best/1/exp, DE/rand-to-best/1/bin** (Equation 9)

$$v = x^G_{i,j} + \lambda \cdot (x^G_{best,j} - x^G_{i,j}) + F \cdot (x^G_{r1,j} - x^G_{r2,j}) \tag{9}$$

- **DE/best/2/exp, DE/best/2/bin** (Equation 10)

$$v = x^G_{best,j} + F \cdot (x^G_{r1,j} + x^G_{r2,j} - x^G_{r3,j} - x^G_{r4,j}) \tag{10}$$

- **DE/rand/2/exp, DE/rand/2/bin** (Equation 11)

$$v = x^G_{r5,j} + F \cdot (x^G_{r1,j} + x^G_{r2,j} - x^G_{r3,j} - x^G_{r4,j}) \tag{11}$$

3.1.4 Genetic Algorithm

Genetic algorithms belongs between most famous evolutionary techniques. Genetic algorithms were introduced by J. Holland. GA is based on evolutionary principle used in nature and described by Ch. Darwin and J. G. Mendel. In classic GA we work with binary representation of individual parameters.

3.1.4.1 Principle

The first population is generated randomly inside specified boundaries. In basic binary representation, it is practically generating 0 or 1 to specified parameters of the individual.

Next is parent selection. This step is on of the key steps of GA. There are 2 basic methods how to select parents. First method is "unit line" (Figure 3) selection where line of size 1 is divided to as many parts as parents where the size of individual parts is dependent on how good is specified individual. This value is obtained by cost functions and then converted to fitness (normalization). Next is generated random number $\in [0, 1]$. Part which contains this random generated number is selected as parent. The same process is repeated for second parent, but it is needed to not to choose same parent twice. In this method is preserved fitness of individuals since each of them has part size equal to its fitness on unit line.

Next method for parents selection is "rank selection". If its not done normalization to interval $[0, 1]$ unit line cannot be used as a parents selection method. In this case, are parents sorted by value from cost function from smallest to largest and parent with the smallest cost value will get number 1 another one will get number 2 and so on. In this method is not preserved fitness of individuals since each individual has the same probability to be taken as a parent.

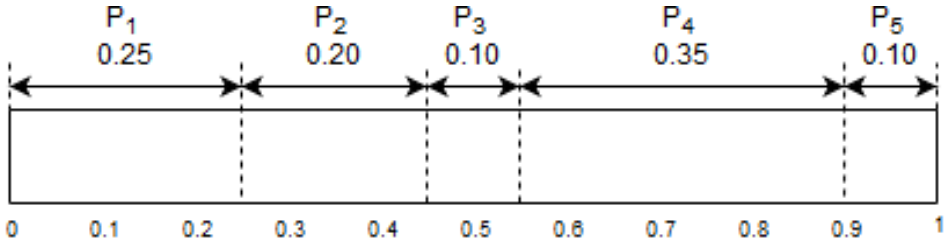

Figure 3: Unit line with parents and its probabilities

As enhancement to these two methods was also introduced "Elitism". Elitism works by choosing best individual in each population and this individual is automatically taken to next population. This step improves process of GA in way that we do not loose best solutions and thus improves performance of GA.

After parents are selected, the next step is crossover. In GA there are three versions of crossover:

- Single-point
- Two-point
- Multi-point

Principle of crossover is the same for all of these versions, but the difference is how many genomes are swapped. On the Figure 4 is showed multi-point crossover. The places where is performed crossover are choosable.

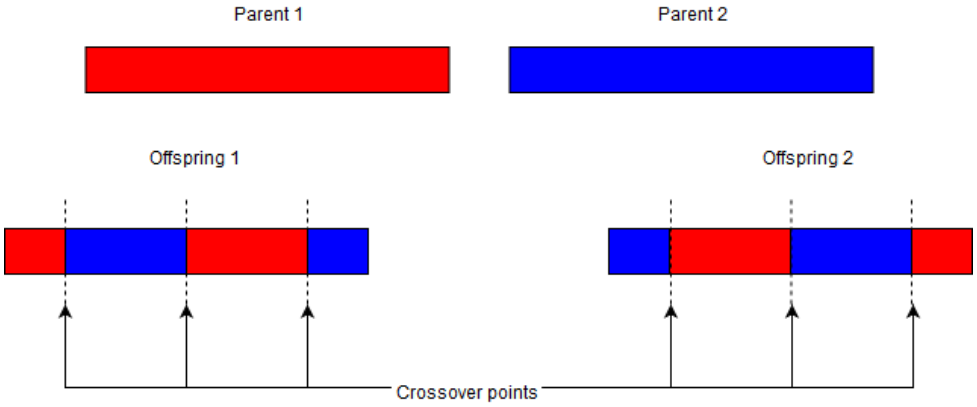

Figure 4: Multi-point crossover GA

After crossover is one final step for individual and it is a mutation. Mutation in GA means inversions of random bits (in binary represented individual). There are also three versions as in crossover: one point, two point and multi point and it means here how many attributes/bits will be inverted.

3.1.4.2 Algorithm

Following pseudo-code (Listing 5) is describing basic version of GA.

```
{
   Input:
   x: first random generated population
   f_cost: cost function returning fitness of actual solution
   populationSize: size of population
}

for i < Generations do
  begin
  while actualNewPopulationSize < populationSize do
    begin
      {make unit line from fitnesses}
      {choose parent_1 individual}
      {choose parent_2 individual}
      {do crossover}
      {do mutation}
      {get fitness of offsprings}
      {put better one to new population}
    end
    {replace old population by new one}
  end
```

Listing 5: Genetic algorithm

3.1.4.3 Adjustable parameters

In GA there are 4 main adjustable parameters:

- **Number of generations**
- **Size of population**
- **Crossover probability** - If crossover probability is 0 %, then all offspring are exact copy of one of its parents. If crossover probability is 100 %, then all offspring are created as crossover of both parents. Recommended value is 80-95 %.
- **Mutation probability** - If mutation probability is 0 %, then the offspring is unchanged. If mutation probability 100 % all attributes/bits are inverted. Mutation was introduced as prevention of stagnation. Recommended value is 0.5-1 %.

3.1.4.4 GA disadvantages

As disadvantages of GA can be accounted the actual problem formulation so GA can solve it. Next disadvantage is parameter adjustment even if there are not many parameters, it may be hard to set right number of generations or population size or the others parameters.

3.1.4.5 Other versions of GA

There are several other version of basic GA, in the following lines is simple description of few of them:

- **Hybrid GA** - This approach is based on searching in space by GA then in local areas are executed local optimization techniques. It can be done by several approaches:
  - GA is running until there are only minor changes and then is executed local optimization technique assuming that algorithm is near global extreme.
  - It is used GA starting population with several local extremes which are founded by random starting individuals.
  - Every x iteration is local optimization technique put its best solution/s as genome/s to population.
- **Messy GA** - This type of GA is designed for faster convergence. In this type of GA every chromosome can be different size and genomes have its own independent position. So every attribute/bit has its value and position. There are two types of chromosomes:
  - Chromosome that have on specific position more values. In this case is there rule of the first - first chosen bit is used.
  - Chromosome is lacking some of the attributes/bits. In this case is used bit from the template. Template is individual with the best solution that was founded yet.

3.1.5 Cost function

In every evolutionary algorithm that was presented before was mention about cost function. In this section, it will be explained what the cost function is.

The cost function can be basically described as function which is formulated by optimization problem. As input have this function parameters for optimization and output is how good solution with given parameters is. In case of optimization it is maximization or minimization of output value.

The cost function can be also illustrated geometrically. Optimization is looking for maximum or minimum on given N+1 dimension surface. Where there are N parameters/dimensions for optimization as input and 1 dimension is for output value from cost function.

If is the algorithm for optimization written only for minimization or maximization and it is needed to optimization in the other way there is easy solution witch rest in the modification

of the cost function to return computed value multiplied by -1. This modification will inverse extremes and required optimization can be processed with the same algorithm without changes in it.

As an example for optimization it is showed Ackley's function on Figures: 5, 6, 7. Parameters for optimization are coordinates X and Y and as output it is coordinate Z. In real situations they aren't known the best parameters values as it is here where it can be seen from images that best parameters values are X = 0 and Y = 0 for minimization.

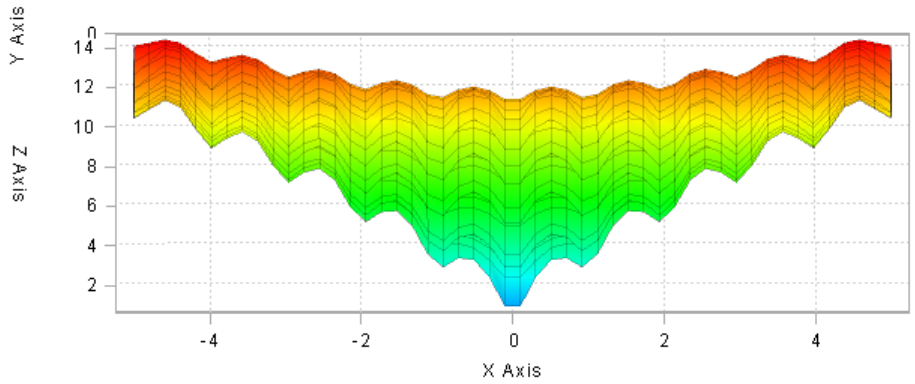

Figure 5: Ackley's function 1

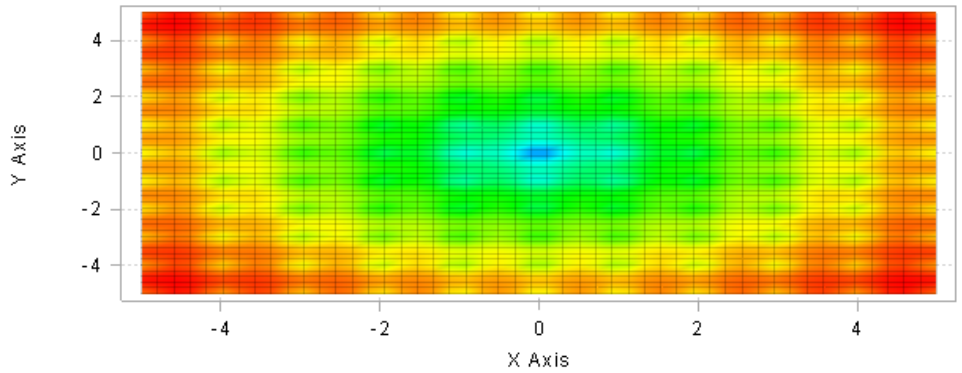

Figure 6: Ackley's function 2

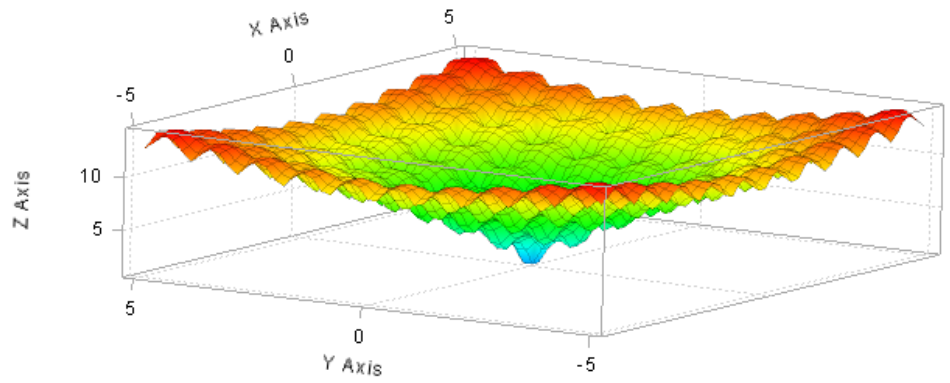

Figure 7: Ackley's function 3

3.2 Types of malware

In this section will be stated various types of malware and their brief description. Malware is abbreviation of malicious software. Malware has a malicious intent. This intent can differ, but in most cases its acts against user will.

3.2.1 Viruses

A computer virus is a code which is recursively spread its copy or slightly changed copy. Viruses infect computer files or elements in OS. The primary purpose of the virus is to spread itself, but it is accompanied by payload most of the time.

3.2.2 Worms

Computer worm is virus which is mainly spreading by networks. Unlike viruses, they don't need host to its function (whole network is their host). There are worms which need user interaction for example worms that are spreading via emails and worms which doesn't need user interaction for example worms that using some exploit to spread itself. Malware can be combined so it can be easily seen malware that acts like worm to spread itself, but as secondary method it can infect files on file system to spread, too.

3.2.2.1 Octopus worm

It is a special type of worm, where the actual worm is divided into parts. Every part is on another location, and together they are performing action as a worm.

3.2.2.2 Rabbit

This type of computer worm is specific by its existence only in one copy in one moment and jumping on network connected computers.

3.2.3 Logic bombs

Logic bomb is type of bug in software, it can be intended or non-intended. It works on success of condition - when this condition is evaluated as true, software performs specific bug - in case of malware it will be probably some malicious code. This method can be even used as defense mechanic of some viruses. Code of logic bomb is mostly hidden in source codes of given software and in this way it can be propagated to next version of software too.

3.2.4 Trojan horses

Trojan horse is a type of malware which trying to pretend to be useful. It tries to take user attention. Trojan horses are often included in software cracks [1]. In another variant trojan horse can contain some new feature included to original exists software, but besides that feature, it contains malicious code too. This new feature can be included in software in two ways: First way rests in available source code for attacker (for example open source projects) and second way rests in repacking software (where is original software packed in new created software with added functionality) or directly modifying software files. Trojan horses are often used to make backdoor in user system or to make some spy activities such as collecting passwords or other sensitive information.

3.2.4.1 Backdoor

Trojan horses are often used to make backdoor on targeted system. These backdoors can be subsequently used to get remote access to infected system. These methods are often based on opening network ports and their subsequent service (for example to access command line). This way is quite good watched by firewalls, but malicious code in trojan horse can try to disable firewall or hide this function to firewall. There is also the possibility to reverse this approach and try connect from infected system to attacker system where attacker listening on specific port - in this way it is much harder for firewall to recognize that it is malicious communication. Another backdoor can for example use email hidden communication with attacker.

3.2.4.2 Sensitive information gathering

Trojan horses are also often used to gather sensitive information. These versions of trojan horses are combined with keyloggers [2] or other spyware which can gather sensitive information, for example, passwords or credit cards number etc.

[1] crack - a method which cracks (break) the software in a way that it can be used illegal (e. g. without serial code)

[2] keylogger - computer software which gathers user pressed keys

3.2.5 First virus generation

First virus generation is kind of "embryo" of computer virus. These first generations are often without hosts files and do not contain some specific features such as the mark of already inflicted file.

3.2.6 Exploits

Exploits are parts of malicious code which is aiming to exploit a specific vulnerability. The purpose of exploit is to run some malicious code or to get access to the target system.

3.2.7 Downloaders

This type of malware has one main purpose - download malicious code. After downloading it, it may be combined with some other exploits to run this malicious code.

3.2.8 Droppers

These are installers of the first virus generation. For example, boot viruses used droppers to get to the boot sector.

3.2.9 Injectors

The injector is a type of malware which injects malicious code. Malicious code can be injected to computer memory or also to network. For example, injector can modify driver for I/O disk operations and then with every operation which is using writing to disk is executed malicious code (e.g. appending virus to file). Network injectors can inject own packets to communication (e.g. inject invalid packet and cause failure).

3.2.10 Auto-Rooters

These are mostly scripts or programs which are used to get administrator rights on targeted systems using a set of exploits.

3.2.11 Virus generation tools

On the internet are available multiple tools to create your own virus without wider knowledge about virus creation. These generators can produce viruses based on given requests. The success of these generators is not 100 % so it can create many non-functional viruses too.

3.2.12 Spam software

This kind of malware is primarily used to generate spam. Spam can be generated not only through the email but also on another communication canals such as various types of messengers etc.

3.2.13 Flooders

These types of malware are usually used to attack network elements. There are many types of flooding methods such as the attack on 3-way handshake on TCP or others. The main purpose of flooder is usually to perform a DoS attack.

3.2.14 Keyloggers

This type of malware is used to gather user pressed keys. This is sensitive information due to chance that we type credentials or other sensitive data (such as credit card numbers) on keyboard. Keyloggers are often combined with other functions such as exploits to make possible send stored data to the attacker.

3.2.15 Rootkits

These are sets of exploits and scripts used by hackers to infect targeted system after gaining administrator access to this system. These are usually modified files and programs which contains another functions and backdoors which can be used lately. We can divide these modified programs (or its parts) to two categories:

- User-mode rootkits - these running in user mode and have limited impact on user system (that does not mean that they are not malicious), but they are easier to get to targeted system
- Kernel-mode rootkits - these are running in kernel mode and have much more possibilities to influence targeted systems - for example these can hide processes (useful when hiding from antivirus) or files, change system registry and much more. But these are more difficult to get on targeted system.

3.3 Virus defense techniques

Over time the authors of viruses founded that if they want to make their work last for a longer time it's needed to defend virus itself against general anti-virus solutions.

3.3.1 Tunneling viruses

Tunneling viruses are viruses which trying to bypass defense systems based on behavior blocking[3]. The way how these viruses are trying bypass defense systems rests in calling directly interrupts so defense monitoring systems does not see these steps.

3.3.1.1 Tracing with debugging

It is a method based on tracing interrupts to its origin. It is using debugging to hang up on interrupts. In this way, whenever is targeted instruction executed is also called debug which leads to trace interrupt address so viruses can access this address directly without calling system API.

3.3.1.2 Code emulation

In this method is virus using code emulator to get entry point address. Code emulator will ensure simulating CPU executing.

3.3.1.3 Communicating with HW

In this technique is virus communicating directly with HW (e.g. HDD) so it's bypassing system API and interrupts. This techniques allows the lowest possible level so there is opportunity to write (in case of HDD) things which will not be allowed by API. But there is big disadvantage too - this technique needs to know how to communicate with specific piece of HW.

3.3.1.4 Using undocumented API

Using undocumented API is a technique which easily bypasses behaviour blockers if it is using APIs which are not known by these blockers. The reason is obvious, if behaviour blocker doesn't know undocumented API function then it doesn't know what to do with it. In this case, can be acquired entry point addresses (and other things too) by these undocumented APIs functions or by combination and tricks with them.

3.3.2 Armored viruses

Armored viruses are viruses whose main purpose is to try as much as possible to hide from or confuse scanners and make harder their analysis for humans and possibly make it impossible for heuristic. In that case, is extended time needed to analyze and so is respond delayed.

3.3.2.1 Defense against disassembling

One of the basic defense mechanism against disassembling is to encode virus data. In this case, it is needed to decode data first and after that can be performed analysis, which means finding decryptor mechanism. However, virus decryptor has to be in decoded state (at least first one in

[3] defense against viruses based on typical virus behavior patterns (e.g. open executable file for writing, modify it, save it)

case of multiple encoding), so virus data can be decoded by the virus itself. Basic encoding can be performed by simple XOR operation with some byte.

Another technique rests in confusing code by writing code ugly and sometimes ineffective (purposefully) - for example when virus want to allocate 100 bytes it can instead of directly allocating 100 bytes do some mathematics operations to calculate 100 bytes than it can copy it somewhere else and copy it back do some loops (very high loop iterations is one of the possible defense method against emulation) and other things which may or may not had nothing to do with actual functionality of virus. This leads to very confusing code which is hard to analyze and thus increasing response time.

Viruses often need some string in code to do their work - for example, email to send sensitive data or names of APIs function to call. These string can be easily readable, and there are many utilities which automatically can extract strings. Because this is not desirable behavior (scanners and heuristic can detect these strings) viruses can use checksums of these strings. For example instead of having a string with open file function name virus can have this string saved as a checksum of this name and use API function which checksum is equal to saved one.

Another technique is compressing the code, and it has one obvious advantage - the virus is after compression smaller. But in addition, the virus has to be decompressed to be readable and can be performed disassembling and analysis. There are many packers which can pack executable files, and they will ensure that when is program launched it is unpacked too.

3.3.2.2 Defense against debugging

Computer viruses are frequently debugged by anti-virus researchers, and thus computer viruses are using defense techniques against debugging too.

Hanging up on debug interruptions is basic method used to prevent debugging. In case that program is not debugged it is harmless, but in case of debugging these interruptions are used by debugger and thus is executed code (it can be even calling another interruption so this interruption will be masked) which is hanged up on these interruptions. Viruses can even use these interruptions for decryptor and again if program isn't debugged decryptor will be executed normally when calling this interruptions, but in case that program is debugged debugger will be using these interruptions (breakpoint for example) even on places that was not intended for virus (e.g. executing decryptor) and thus virus will not work.

Another technique for defense against debugging use checksums to control integrity of itself. If is program debugged there are inserted breakpoints to code and thus integrity check will fail. These technique can be enhanced to error-correcting code so it will effectively remove breakpoints.

If the virus is monitoring stack when the program is launched it is possible for the virus to detect debugger. The debugger is using the stack to store tracing records, and thus virus monitoring stack can find the debuggers records and recognize debugger.

Another methods for detecting debuggers are quite simple. Viruses can find that there is installed debugger on OS by searching in OS registry or alternatively virus can scan memory for signs of running debugger etc.

Interesting method to block debugger after it is detected is to block keyboard so user cannot use keyboard shortcuts to tracing. Another interesting method was used by virus "Cryptor" which used keyboard buffer for storing cipher keys and thus when was virus debugged and user was using keyboard to shortcuts (e.g. next step) cipher keys were rewritten.

Using exception handling method as a defense against debugging is a method which is based on the fact that debuggers may not have implemented all exceptions and so debugger will forward exception handling to OS.

3.3.2.3 Defense against heuristic analysis

There are two types of heuristic - static and dynamic. Static heuristic analysis is analysing file format, and fragmentation in code such example can be appending virus. In case of appending virus, it is placed jump somewhere near the start of the executable file. This jump pointing somewhere near the end of the file, and so this behavior can be marked as suspicious by static heuristic. Dynamic heuristic analysis is working with simulation and detection of suspicious activities which occur on-the-fly. There are several ways how viruses are trying to confuse heuristic analysis.

One of the methods which used as a defense against heuristic, is connecting to more than 1 section in PE file. In this case, is virus spread amongst the whole file, and it is harder for heuristic scanner to find that the file is infected by the virus.

Another anti-heuristic technique is to encode original file header so heuristic will be confused by this file.

Next anti-heuristic method is based on overwriting blank part of the first section with the jump to start of the virus which can be located anywhere else.

Another anti-heuristic technique is based on shifting sections in PE. The whole virus is placed in the code section, but most viruses aren't so small that they can fit the whole virus in this (there is need to be enough space there) section so virus will place itself to code section and shift all others sections so for static heuristic it looks like ordinary file.

Next method is based on the packing code section and then appending to this section. In this case, virus doesn't need to shift other sections to fit there. After virus activation virus unpack code section so the origin code section can be executed fine.

Another technique is technique using entry point obstruction. In this case virus do not get control directly by overwriting entry point to itself, but virus is trying to overwrite/add instruction to jump to yourself somewhere in file.

A similar technique to the previous one is technique when the entry point of virus is selected randomly in code section. In this technique virus overwrite jump or call instruction somewhere near original file entry point so virus take control after this instruction is executed.

Next technique is using compiler alignment. The virus will divide itself into parts and fit itself to these alignment areas).

Another anti-heuristic technique is a technique where virus does not mark any section as writable and execute itself on the stack.

Next technique which viruses are using to confuse heuristic scanners is a method where the virus is trying to found a collision in checksum algorithms, so it can have the same checksum after infecting.

Another anti-heuristic method is to restrict infection of header because infection of the header is often suspicious for heuristic.

Limitation of CALL-POP instructions. These instructions are used by viruses to detect memory address location where is the virus itself located. This technique is quite simple and effective, but for heuristic scanners, it isn't a big problem to evaluate this behavior as suspicious.

As already mentioned in defense against disassembling, method using checksums of APIs functions name is used as defense against heuristic too. So heuristic scanners cannot easily distinguish these strings.

3.3.2.4 Defense against emulation

Emulation is used for exploration of how the virus works on-the-fly. In next paragraphs will be described some of the techniques used by viruses for defense against it.

Using instructions that are not implemented in emulator. In the past emulators didn't have implemented instructions for FPU and MMX (multimedia instructions), so viruses used them in its code. These instructions then cannot be emulated. Nowadays anti-viruses have emulators with these instructions implemented.

Another method using by the viruses to disable emulators is using SEH. In this case, viruses try to raise an exception which handling is not implemented in the emulator. So emulator may not get to the virus code if the exception is raised sooner.

Next method is using "randomly" executing virus code. What it means is that virus code is not executed whenever is host file launched. So virus hosts file can be emulated when the virus will not be executed at all.

Another method is based on using undocumented processor instructions - it is the similar case as when older emulators didn't have implemented FPU and MMX. When emulator doesn't know these instructions, it can't emulate them.

Next method is using brute force to decode itself. In this case virus doesn't contain key to decode its body and instead of key it use brute force to decode. For emulators it uses many iterations for decoding and thus with combination of emulator is not as fast as running program directly it can take too much time to accomplish it (users don't want to wait too much).

Another method which is used as a defense against emulations is a method based on multi-threading. In this case, viruses are trying to use multi-threading and hope that emulator will not have multi threading or APIs associated with it implemented.

Using long loops is a similar anti-emulation method as method described above which uses brute force to decode itself. But in this case, it does not have to be decoding which causes a long loop, but anything else. It may be a thing which is needed for running the virus, but it may be not - for example, some calculations which are meaningless in term of successful virus execution.

3.3.3 Retroviruses

Retroviruses are trying to evade or to disable anti-viruses, firewall or other security systems.

Retroviruses can be used for disabling security systems and making a path for viruses which will be normally detected by these safety systems.

In the following list are stated some of the techniques used by retroviruses:

- Disabling and/or removing security systems from memory or file system. (anti-viruses, firewalls etc.)
- Removing or modifying files which are used for integrity check.
- Attack anti-virus so anti-virus itself can do malicious activity - for example removing other security systems.
- Try to escape while virus is scanned (e.g. attack the scanner and/or infect another files with scanner)
- Preventing to download virus databases for anti-viruses or updates (the same applies to other security systems or even for OS security updates).

3.4 Experiments

Not many experiments were done with the use of evolutionary malware. Following lines will describe 2 of them.

3.4.1 Evolvable malware

Experiment[2] done by S. Nooren, S. Murtaza, M. Z. Shafiq and M. Farooq is following up creating new viruses using the genetic algorithm and known virus family called Bagle. Authors made a simple framework which consists of 3 modules:

- Code analyzer - it ensures generating high-level genotype of the virus from machine code
- Genetic algorithm
- Code generator - it ensures code generation from newly created genotype

The Figure 8 describe architecture of evolution malware framework.

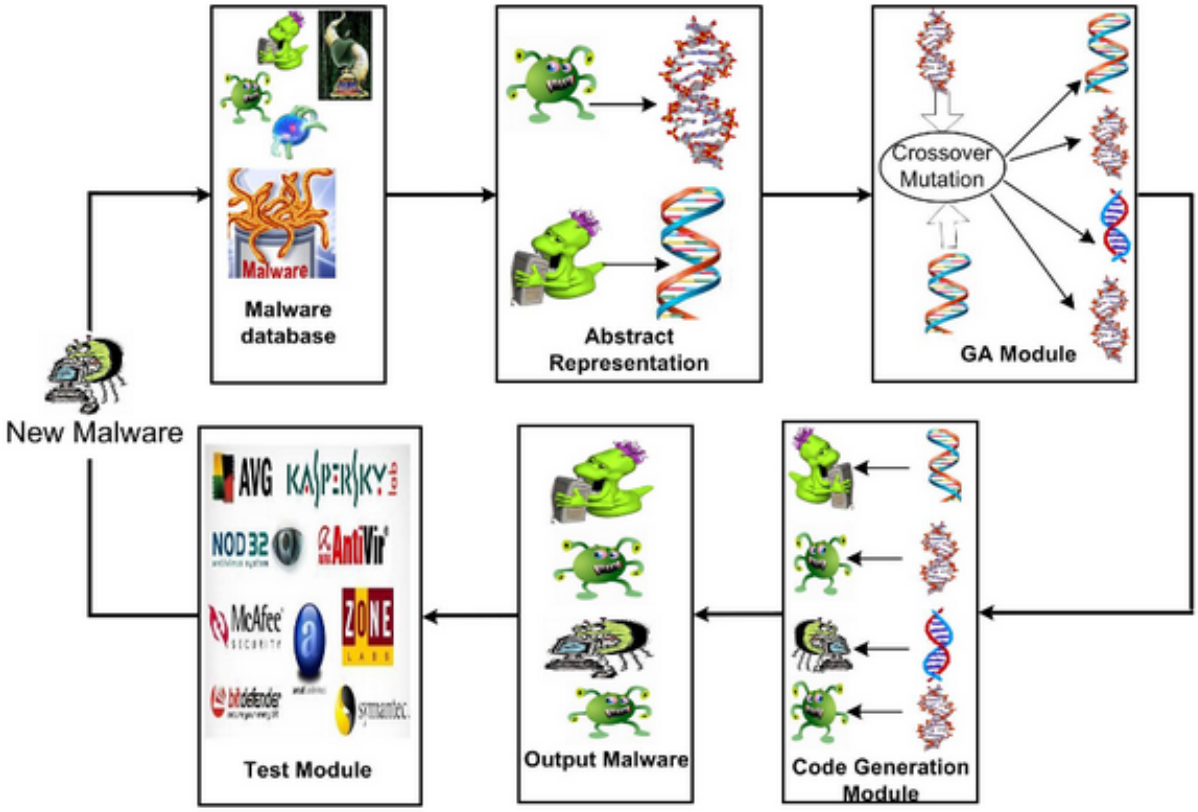

Figure 8: Evolution malware framework

Source: [2]

3.4.1.1 Bagle

Bagle is a mass-mailing worm. There are several versions of Bagle. Some versions of Bagle contains its own SMTP engine. Bagle also contains backdoors. For example, it can be opening TCP ports and listening on them and copying itself to System directory.

3.4.1.2 Abstract representation

The Table 1 shows Bagle abstract representation with all features, its description and example values. From the abstract representation authors created "gene library" which contains all possible feature values for all features. These features are then used to form individual for GA.

3.4.1.3 Experimental setup

In the experimental setup (Figure 9), authors work with 15 unique variants of Bagle from various sources.

At first were Bagle samples divided to training and testing categories. Fitness function was formed as similarity of chromosomes against chromosomes in training category then is fitness normalized by assigning weight to each gene of representation. Offsprings for next generation

Table 1: Abstract representation of Bagle

Feature	Description	Examples
Date	The date checked by Bagle to (de-)activate its process	28 January, 2004
Application	The applications used to conceal Bagle	calc.exe, notepad.exe, sndrec32.exe
Port Number	Port opened by Bagle to send or receive commands	2475, 6777, 2556
Attachment	Name of the attachment used by Bagle	Random characters
Attachment Extension	It specifies the extension of the attachment	.rar, .exe, .pif, .zip
Websites	Bagle contact the websites to inform about the infection	http://www.it-msc.de/1.php, http://www.getyourfree.net/1.php
Domain	Bagle ignores to email itself to the domains specified	@hotmail.com, @msn.com
Email Body	Contains the email body of Bagle	Test=), YoursID <Random Characters>
Email Subject	Specifies the subject of the email	Hi, Subject:ID <Random Characters>
Registry Variable	Contains the name of registry variables used by Bagle	au.exe, d3dupdate.exe
Virus Name	Name of the Bagle shown in the task manager	bbeagle.exe, au.exe, readme.exe
File Extension	File extensions to be searched in the fixed directories	.wab, .txt, .htm, .php
Process Terminated	Processes terminated by Bagle	atupdater.exe, aupdate.exe
P2P Propagation	Names used by Bagle to copy itself to peer computers	ACDSee 9.exe, Ahead Nero 7.exe

Source: [2]

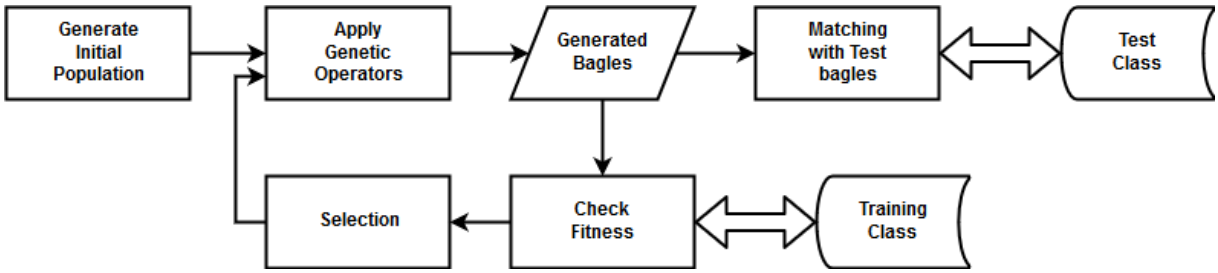

Figure 9: Experimental setup

Source: [2]

are selected based on their fitness. Thus exact match of evolved Bagle individual is equal to fitness = 1. There can happen 3 conclusions when individual is generated and its fitness $\neq 1$. These conclusions are:

- Individual is from test category
- Individual is new Bagle virus
- Individual is not a Bagle virus

Genetic algorithms parameters are seen in Table 2. For selection were used multiple methods: roulette wheel, rank, tournament. Crossover methods used in GA were one-point, two-point and uniform.

Table 2: Parameters configuration of GA - Bagle experiment

Crossover probability	0.75
Mutation probability	0.005
Population size	500

Source: [2]

3.4.1.4 Results

Authors show results based on how close was generated individuals fitness to fitness of specific Bagle types and show which type of crossover and selection was used to achieve the best fitness. Next author tested sample evolved malware by commercial antivirus software AVG and Symantec. Detection rates are shown in Table 3. The interesting thing is that Symantec antivirus has identified in 14.64 % cases from 41.1 % (total detected) virus as "W32.Sality.AE" which was not included nor in training neither in the testing set.

Table 3: Detection rates of evolved malware

	AVG [%]	**Symantec** [%]
Detected	98.98	41.1
Undetected	1.02	58.9

Source: [2]

This means that it is possible to evolve new malware from other samples as authors stated: "*These results prove our claim that the proof-of-concept malware evolution engine has successfully evolved unseen malware.*"

3.4.2 Towards Automated Malware Creation

In the experiment done by A. Cani, M. Gaudesi, E. Sanchez, G. Squillero and A. Tonda were used an evolutionary algorithm (μGP - `http://ugp3.sourceforge.net/`) to determine the optimal position for hiding malicious code inside an existing executable and to code generation.

3.4.2.1 Code generation

For code generation authors used already mentioned EA - μGP and inserted to initial population code of the virus "Timid". Timid is simple file infector virus which whenever is executed infected ".COM" file it copies its code to other uninfected ".COM" file within the same directory. Framework schema is shown in Figure 10.

Authors used assembly compiler and DOS scripts which kills individuals process if they run more than 5 seconds (to ensure killing infinite loops).

Authors then choose four freeware antiviruses which all of them has to have heuristic only detection which is configurable and fast scanning. Database driven detection is excluded due to newly generated malware will not be in the database, and it is time-consuming procedure.

Next authors used 5 ".COM" files from the "system32" folder and check them with md5 function to get their hash, so integrity check can be done subsequently.

The initial population of each run of the framework contains only one individual who represents the original code of Timid. Parameters are set $\mu = 10, \lambda = 8$ for each framework run.

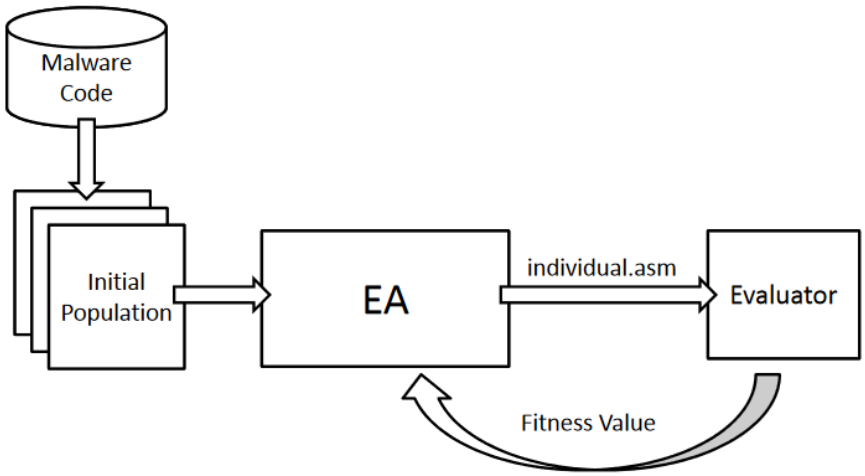

Figure 10: Framework schema for code generation

Source: [3]

3.4.2.2 Code generation results

Max fitness value was reached 96 times of 100 which means that 96 newly created malware are behaving like the original one, are compiled without errors and executed successfully. All four free antiviruses don't detect this newly generated malware by its heuristic. Framework runs are in an average terminated in the 6th generation, and the standard deviation is 2.5.

3.4.2.3 Code integration

Authors selected two executable files to perform code injection. First one is the program which is used to split files to smaller parts and also join them to origin file. Its executable is named SPLIT.EXE with size 46.5 kB (`http://www.iacosoft.com/home/split.txt`). The second one is data recovery software TESTDISK.EXE with size 315.2 kB (`http://www.cgsecurity.org/wiki/TestDisk`). Framework schema for code injection is shown in Figure 11.

Authors define two types of zones where to inject code:

- **Type I** - zones are almost always skipped during regular execution
- **Type II** - zones that are usually not processed in normal flow and often appear after the end of the main function

According to authors, it is rationale to use Type I zones to overwriting with a call to call malicious code and Type II zones to store actual malicious code.

An evolutionary algorithm is there used to compute parameters offset and size. Where offset is the offset from the beginning of the compiled code in bytes and size is the size of the part in bytes.

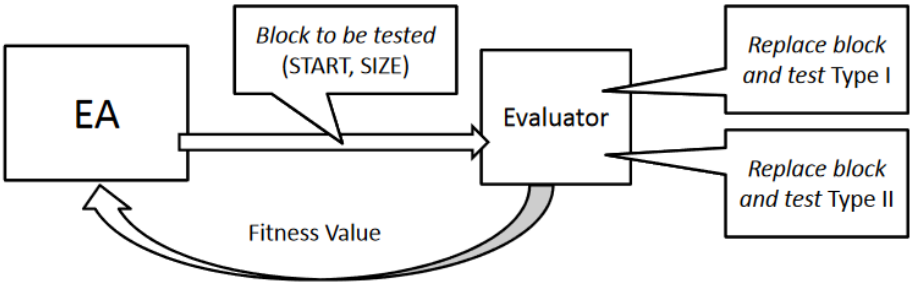

Figure 11: Framework schema for code injection

Source: [3]

3.4.2.4 Code integration results

For executable SPLIT.EXE were parameters set to $\mu = 100, \lambda = 30$, offset $\in (0, 43000)$ and size $\in (12, 1000)$. Results are seen in Table 4.

Table 4: Code integration experiments - SPLIT.EXE results

SPLIT.EXE	
Evaluations	300
Type I (zones found)	1
Type I (largest zone)	334
Type II (zones found)	32
Type II (largest zone)	1511

Source: [3]

For executable TESTDISK.EXE were parameters set to $\mu = 100, \lambda = 30$ and size $\in (12, 1000)$. Results are seen in Table 5.

3.4.2.5 Conclusion

Authors showed how to improve malware to escape detection via an evolutionary algorithm in the first experiment. In the second experiment authors showed how to improve searching for weak points in application to perform code injection.

Table 5: Code integration experiments - TESTDISK.EXE results

TESTDISK.EXE			
Offset	(0,43000)	(0,10000)	(0,2000)
Evaluations	15000	2000	300
Type I (zones found)	-	1	1
Type I (largest zone)	-	33	25
Type II (zones found)	3	4	3
Type II (largest zone)	179	167	183

Source: [3]

4 Practical part

This section will be divided into three subsections. The first one is a description of written virus and its modules including how these modules work. The second one is user manual on how to setup the virus to use. The last one is detection testing of the virus.

4.1 Virus description

The virus itself is divided into three main parts:

- Main program - the part which caters for virus running
- Modules - this part contains modules that are used by the evolution process to create new a virus
- Evolution - the part which caters for virus evolution via evolutionary algorithms and cost functions

4.1.1 Main program

In this section is described work of virus execution in a nutshell. A more detailed description of individual modules is in the modules section.

Main program consists of "virusSize" (Listing 6) variable with set value. This value is changed after compilation to get exact size of compiled virus this is described more in modules section in infection part.

```
int virusSize = 1234567890;
```

Listing 6: virusSize variable

Then there is a try block where takes place whole virus execution. Raised exceptions are handled, and then logged to file ("virus-exceptions.txt") on active windows user desktop.

At first, there are initialized variables for pseudo-random number generation, an instance of actual virus and an instance of the compiler.

After initialization is reconstructed original file (via infector module) that user means to launch with parameters for launch forwarded.

After reconstructing phase starts virus execution. At first, there is a logical bomb which conditions must be met or virus will not be executed.

If logical bomb condition is met then is starting the evolution of the new virus. It is chosen fitness function and evolutionary algorithm with computed parameters which will be responsible for virus evolution.

Evolved malware is then forwarded to compiler module which ensures its compilation and loading it to the byte array.

After compiling and loading virus to the byte array there are searched victims for new virus by currently executing virus via "VictimSearcher" module.

After victims are founded, there is forwarded byte array of a new virus and number of victims to infect parameter to "infector" module of the currently executed virus. Then the current virus will start to infect its victims.

After infection of victims, there is part with logging data about the current virus, victims, newly evolved virus etc. This part, of course, exists only due to the experimental nature of this master thesis. The true virus will probably not have this section or will be gathering other information that is more relevant for it.

Then is executed payload module of the currently executed virus.

4.1.2 Modules

Modules are divided to 5 categories:

- Compilers
- Infectors
- LogicBombs
- Payloads
- VictimSearchers

Each category consists of one or more modules that do their work that is specified by its category. Each module in the category have to implement the interface of appropriate category, e.g. modules in category Compilers have to implement "ICompile" interface. These modules are then used by evolutionary algorithms to create new malware.

4.1.2.1 Compilers

The compiler module is responsible for compiling new virus individual from the given main structure. Compilers modules has to implement ICompile interface (Listing 7).

```
public interface ICompile
{
    byte[] Compile(string main);
}
```

Listing 7: ICompile interface

In this work was written only one compiler class ("Compiler.cs) and one helper compiler class ("HelperCompiler.cs").

Helper compiler class caters about finding the path to "csc.exe" which is C# compiler provided by .NET Framework. This goal is achieved by looking for the correct key in the registry. If the path to "csc.exe" is founded it is returned.

Compiler class (Compiler.cs) method "Compile" first gets the path to "csc.exe" from helper class and if the path is founded it continues to the next step.

Again compiler work is contained in a try block to handle possible raised exceptions.

At first compiler tries to create temp directory in Windows temp folder. Name of this directory is number from 10000 to 10000000. Compiler generates random number in this range and if there is a directory with the same name it is generated a new random number. This process continues until the directory is created.

After temp directory for the compiler is created. There is saved and extracted zip archive ("virus.zip") with the source code of modules, dynamic linked libraries which allows using compression + PowerShell in virus and "files.txt" file which contains relative paths to all files that are needed for compilation.

After that, compiled resources are extracted from currently running assembly. This extraction is performed by binary reading actual assembly and searching for saved byte array pattern. This byte array pattern is saved in compiler class in reverse order, because otherwise there will be always 2 matches of this pattern (1 desired result + 1 pattern result). When this pattern is founded specified amount of bytes ("resourcesSize" variable)(Listing 8) is copied and saved as "virus.Properties.Resources.resources" to temp directory. This *.resources file is compiled resource file that contains zip archive "virus.zip". The "resourcesSize" integer variable has to have value which is equal or a few bytes greater than size of compiled resources file to ensure that resource file can be used in compilation of virus.

```
int resourcesSize = 52000; //a few bytes more to eliminate compression
    fluctuation
```

Listing 8: resourcesSize variable

After all required files are extracted compiler creates file "Program.cs" with the structure which was given to compiler as a parameter.

At this moment all files needed for compilation of new virus are prepared. Then is created a string with all required parameters and with all relative paths to files to compile.

This string is then together with the path to "csc.exe" passed to "ProcessStartInfo", and this process ("csc.exe") is started in a hidden window.

After compilation ends "virus.exe" is loaded to memory, compiler perform cleanup of temp folder and loaded virus is returned.

4.1.2.2 Infectors

Infectors are responsible for infection and reconstruction of the host. The interface that infectors have to implement is shown in Listing 9.

```
public interface IInfector
{
    int InfectorID { get; }
    List<string> Infect(HashSet<string> filenames, byte[] virus, int
        numberOfFilesToInfect);
    void Reconstruct(int virusSize, string[] args);
}
```

Listing 9: IInfector interface

In this work was written prepend infector class ("PrependInfector.cs") and one helper infector class ("HelperInfector.cs").

HelperInfector class caters about PreInfectRoutine, combining two bytes array and selecting files for infection if there are more files to infect available than is requested.

PreInfectRoutine (Listing 10) is a method which gets as parameter virus in a byte array. The method then performs requested changes on a virus byte array. These changes are creating infection mark which is created by inserting by "0x6E" ('n' in ASCII) to position 89. This is the position where is in EXE files placed the text "This program cannot be run in DOS mode." or other variations and changes this text to "This progra**n** cannot be run in DOS mode.".

After placing infection mark, it is computed the length of virus byte array, and it is searched bytes that locates the reloc section in PE file. Then is again searching for the pattern for the integer value of "1234567890" which is the value of "virusSize" variable in the main function in "Program.cs". These bytes are then replaced by bytes from integer variable that represents actual virus length. Patterns are again saved in routine in reversed order to get exact match results when searching in a virus byte array.

```
public static void PreInfectRoutine(byte[] virus)
{
    virus[89] = 0x6E; //infection mark

    byte[] relocPattern = new byte[] { 0x63, 0x6F, 0x6C, 0x65, 0x72, 0x2E }; //
        reversed
    byte[] sizePattern = new byte[] { 0x49, 0x96, 0x02, 0xD2 }; // reversed,
        int 1234567890
```

```
    byte[] newSize = BitConverter.GetBytes(virus.Length);

    int relocIndex = virus.FirstIndexOf(relocPattern.Reverse().ToArray());
    int sizeIndex = virus.FirstIndexOf(sizePattern.Reverse().ToArray(),
        relocIndex);

    for (int i = sizeIndex; i < sizeIndex + 4; i++)
        virus[i] = newSize[i % sizeIndex];
}
```

Listing 10: PreInfectRoutine method

In PrependInfector class there are two methods: Infect and Reconstruct and one property: InfectorID.

The infect method ensures infecting files, it has parameters: filenames - full path to potential victims, virus in a byte array, number of files to infect.

At first "Infect" method calls function from helper class to get exact HashSet of files to infect. Then is executed "PreInfectRoutine" on a virus byte array. After that are all files infected by prepend virus bytes to host file bytes. The old host file is deleted and replaced by infected host file.

Reconstruct method ensures reconstructing and launching the original host file for the user with forwarded arguments.

At first "Reconstruct" method get the location of which is currently launched assembly launched. After that, it reads from this location itself and creates string arguments for the launching application. Then it checks if the reconstructed file is already existing if so it is launched with passed parameters. If not it is calculated victim size and it is created origin host file with '.' character on the start of the name and with file attribute hidden set. Then is launched this reconstructed host file with passed arguments.

4.1.2.3 LogicBombs

Logic bombs are responsible for virus triggering. When the condition of the logic bomb is met the virus code will be executed. The interface that logic bombs have to implement is shown in Listing 11.

```
    public interface ILogicBomb
    {
        int LogicBombID { get; }
        bool Execute(object o);
    }
```

Listing 11: ILogicBomb interface

Object "o" in the "Execute" method can be any object, some of the logic bombs working with that object. In this work is that object used in Program.cs main method and it is string consisted of 'a' characters with random length in range 1 to 100.

There are 7 logic bombs implemented in this work:

- MagicianLogicBomb
- Magician2LogicBomb
- ObjLengthLogicBomb
- ObjLength2LogicBomb
- RandomLogicBomb
- Time1LogicBomb
- Time2LogicBomb

MagicianLogicBomb works on math trick with number 1089. This trick using a three-digit number which digits are different from each other. From selected number is subtracted the number that is created by reversing order of digit from origin number. Then is taken the absolute value of this result as the new result. To this new result is added number that is again created by reversing digit order and the result of this is 1089. There is $9 \times 9 \times 8$ numbers that will give this result. So it is $\frac{9 \times 9 \times 8}{900} = 0.72 => 72\%$ chance of getting this result. In practical testing on 1×10^6 iterations were results 0.71 ± 0.01.

Magician2LogicBomb class works on another math trick. In this trick it is randomly selected integer number from interval $< 1; 6 >$ and then this number is multiplied in this order by numbers 9, 111, 1001 then the result is divided by 7. The result will contain digits 1, 2, 4, 5, 7, 8. In this class is as input chosen random number from interval $< 1; 9 >$ so the trigger probability is $\frac{6}{9} = \frac{2}{3} \approx 0.66 => 66\%$.

ObjLengthLogicBomb works on object.ToString() length, when length modulo 2 = 0 then logic bomb triggers.

ObjLength2LogicBomb works on object.ToString() length, when length modulo 3 != 0 then logic bomb triggers.

RandomLogicBomb works on generating random double in the interval $< 0; 1.0)$ and when the number is lesser than 0.65 it triggers.

Time1LogicBomb triggers in dependency on actual Unix time seconds. If this number modulo 2 != 0 logic bomb triggers.

Time1LogicBomb triggers in dependency on actual Unix time seconds. If this number modulo 3 != 0 logic bomb triggers.

4.1.2.4 Payloads

Payloads are responsible for performing malicious activity. Every payload has to implement interface shown in Listing 12.

```
public interface IPayload
{
    int PayloadID { get; }
    void Execute();
}
```

Listing 12: IPayload interface

There are five payloads in this work, but all payloads are just message boxes with the following strings:

- "Cheers!"
- "Gimmie cookie!"
- "Good <morning/afternoon/evening>, <windows user name>."
- "I got you!"
- "You are infected! <actual datetime>"

4.1.2.5 VictimSearchers

VictimSearchers are responsible for looking for victims. Every VictimSearcher has to implement interface shown in Listing 13.

```
public interface IVictimSearcher
{
    int SearcherID { get; }
    HashSet<string> GetFiles();
}
```

Listing 13: IVictimSearcher interface

There are 5 VictimSearchers and one helper class in this work.
Helper class is responsible for 3 main things:

- Getting file location from the given link (*.lnk) - for this purpose is used library that allows using of PowerShell
- Recursive searching in directories with given depth and start path

- Checking if the file can be infected - checking infection mark and file attributes

Among actual VictimSearchers belongs these five VictimSearchers:

- DesktopSearcher
- StartMenuSearcher
- TaskBarSearcher
- TreeSearcher
- UserAssistSearcher

All VictimSearchers logic is surrounded witch try block. Exceptions are then handled and logged to "virus-exception.txt" file on user desktop.

DesktopSearcher is using "Environment" variable to get the path to actual user Desktop folder then starts recursively searching for victims on Desktop and its sub-directories.

StartMenuSearcher is using "Environment" variable to get the path to actual user "application data" folder and then is this path concatenated with the string "Microsoft\Windows\ Menu\" result represents the path to current user Start Menu folder. Then StartMenuSearcher starts recursively searching for victims there and in its sub-directories.

TaskBarSearcher is using "Environment" variable to get the path to actual user "application data" folder and then is this path concatenated with the string "Microsoft\Internet Explorer\Quick Launch\User Pinned\TaskBar\" result represents the path to current user pinned items on Task Bar. Then TaskBarSearcher starts searching for victims there.

TreeSearcher gets the actual launched assembly path and then combine it with ".." string which means to get one directory upwards. From there searcher starts searching recursively for victims including sub-directories.

UserAssistSearcher is using registry entries of windows where are stored launched applications. The registry path in which searcher looking is
"Software\Microsoft\Windows\CurrentVersion\Explorer\UserAssist\" + one of the nodes listed below + "\Count"

- "CEBFF5CD-ACE2-4F4F-9178-9926F41749EA"
- "F4E57C4B-2036-45F0-A9AB-443BCFE33D9F"

Entries there are encoded with rot13 encryption, so for getting the path to the application it is needed to decrypt them. Also for getting full path there is need to assign GUIDs to correct folder, because entries are saved in form of <rot13 encrypted GUID>+<encrypted rot13 relative path>. So when searcher passing nodes it is needed to decrypt whole string entry and then replace known GUID with correct folder. Correct known GUIDs for Windows 7 and newer are:

- "1AC14E77-02E7-4E5D-B744-2EB1AE5198B7"
  "C:\Windows\System32"
- "6D809377-6AF0-444B-8957-A3773F02200E"
  "C:\Program Files"
- "7C5A40EF-A0FB-4BFC-874A-C0F2E0B9FA8E"
  "C:\Program Files (x86)"
- "F38BF404-1D43-42F2-9305-67DE0B28FC23"
  "C:\Windows"
- "0139D44E-6AFE-49F2-8690-3DAFCAE6FFB8"
  "C:\ProgramData\Microsoft\Windows\Start Menu\Programs"
- "9E3995AB-1F9C-4F13-B827-48B24B6C7174"
  "<current user appdata>\Microsoft\Internet Explorer\Quick Launch\User Pinned"
- "A77F5D77-2E2B-44C3-A6A2-ABA601054A51"
  "<current user appdata>\Microsoft\Windows\Start Menu\Programs"
- "D65231B0-B2F1-4857-A4CE-A8E7C6EA7D27"
  "C:\Windows\SysWOW64"

After decryption of entries searcher can use CanInfect method from the helper and then decide if the entry will be selected as a victim.

4.1.3 Evolution

Evolution prosecutes 3 main parts:

- Evolutionary algorithms - implementation of individual evolutionary algorithms
- Individuals - virus individuals which are used in cost function and evolutionary algorithms
- Fitness functions

4.1.3.1 Evolutionary algorithms

Evolutionary algorithms part consists of one helper class and four classes with implementation of actual evolutionary algorithms.

All evolutionary algorithms has to implement interface shown in Listing 14.

```
interface IEvolution
{
    Individual Execute();
}
```

Listing 14: IEvolution interface

The helper class is responsible for generating random population, generating random individual and for converting ranges - which is needed for some of the implemented algorithms.

Implemented algorithms are listed below. How algorithms work is showed in the theoretical part of this master thesis. All these algorithms are solving minimization task.

- Differential evolution in version DE/rand/1/bin
- Genetic Algorithm
- Particle Swarm with inertia implementation
- Self Organized Migration Algorithm (SOMA) in version All to One

4.1.3.2 Individuals

Individuals part consists of helper class and Individual which inherits from RawIndividual.

The helper class is responsible for getting virus structure and list of all specified modules (all infectors, all logic bombs etc.). Also in this class is "enum" with the structure of the virus. The structure of the virus is shown in Table 6.

Table 6: RawIndividual virus structure in integer array

int[4] structure							
0		1		2		3	
LogicBombs		**VictimSearchers**		**Infectors**		**Payloads**	
ID	Name	ID	Name	ID	Name	ID	Name
0	Magician2LogicBomb	0	DesktopSearcher	0	PrependInfector	0	CheersPayload
1	MagicianLogicBomb	1	StartMenuSearcher			1	CookiePayload
2	ObjLength2LogicBomb	2	TaskBarSearcher			2	GoodXYPayload
3	ObjLengthLogicBomb	3	TreeSearcher			3	IGotYouPayload
4	RandomLogicBomb	4	UserAssistSearcher			4	YRInfectedPayload
5	Time1LogicBomb						
6	Time2LogicBomb						

RawIndividual class contains several constructors for creating RawIndividual. Every RawIndividual consists of LogicBomb, VictimSearcher, Infector, Payload. From input parameters are filled all properties and it is created structure in the integer array format too. Another thing which RawIndividual caters about is creating the main program file. This file is created in dependency of which modules are used in instance of the individual.

Individual is class which inherits from RawIndividual, and it adds fitness property which is in use by evolutionary algorithms and fitness function respectively.

4.1.3.3 Fitness functions

The fitness functions are responsible for measuring how "good" is the given individual. All fitness functions have to implement interface shown in Listing 15.

```
public interface IFitnessFunction
{
    double GetFitness(RawIndividual individual);
}
```

Listing 15: IFitnessFunction interface

In this work is implemented one fitness function (called DistanceFitnessFunction) based on distance or difference of actual individual from its ancestor. There is a distance matrix for each module (infector, logic bomb, victim searcher, payload) that is used in RawIndividual. This matrix contains distances of module individuals from each other. Metric for computing distance between modules is based on the number of similar programming elements used in module and can be viewed in "distanceMatrix.xlsx" attachment of this work.

Fitness is computed based on the distance of module individuals (current and ancestor) from each other and added together for all modules. Then is fitness multiplied by -1, because of algorithms are implemented for solving minimization.

4.2 User manual

Deployment of virus developed in this work is not so simple as deploying a classic application. Deployment is performed in the following steps:

1. Creating virus.zip archive. This archive consists of folders Evolution, Extension, Modules contained in visual studio solution in the sub-folder virus. These folders (Figure 12) contains sub-folders and files with source code. Whenever is the source code of virus changed it is needed to copy and overwrite these folders to "virus.zip" archive to ensure that next iterations of the virus have updated code. Next files that are contained in the "virus.zip" archive are:

    - "Resources.Designer.cs" - file generated by Visual Studio that contains information about resources. If a new resource is added to the solution, it is needed to overwrite this file too. This file is located in Properties sub-folder of virus starting solution folder.
    - "Interop.IWshRuntimeLibrary.dll" - library that is allowing using PowerShell in virus
    - "files.txt" - the file that contains relative path (from "files.txt" location) to all source code files including "Resources.Designer.cs"

File virus.zip is available as an attachment of this work too, but if there are made changes in the code, it is needed to overwrite respective folders and files too.

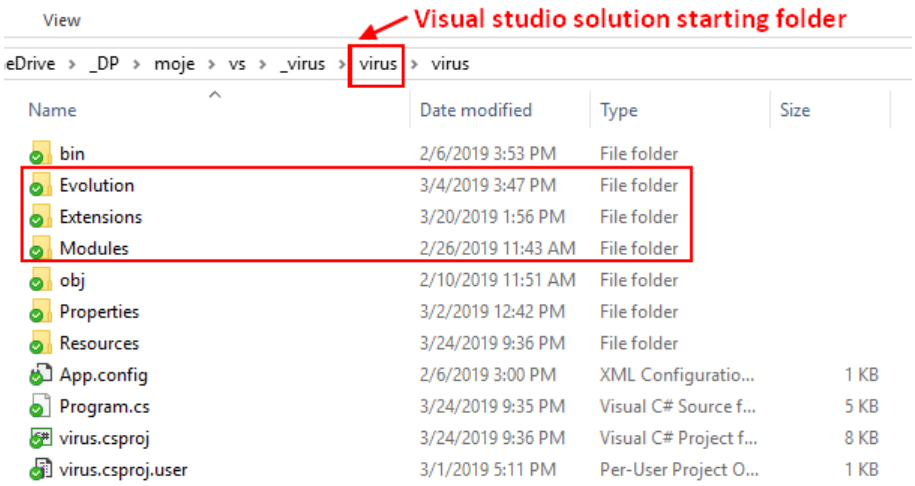

Figure 12: Evolution, Extension, Modules folders

2. Placing virus.zip to solution resources (Figure 13).

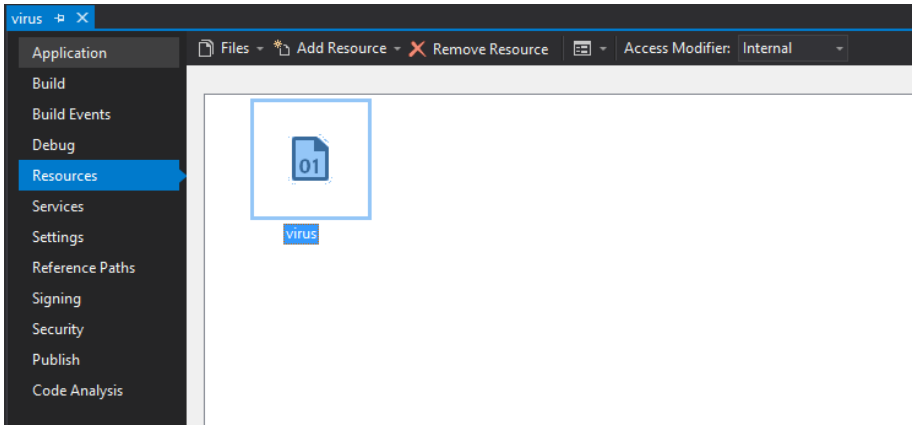

Figure 13: Solution resources

3. Delete from obj folder (Realease/Debug - it depends on what type we are building) file "virus.Properties.Resources.resources".

4. Build solution.

5. Check what size has newly created file "virus.Properties.Resources.resources" in obj folder. If file size is greater then value of variable "resourcesSize" (Figure 14) in "Modules/Modules/Compilers/Compiler.cs" file then it is needed to adjust value of this variable to be a few bytes greater than size of "virus.Properties.Resources.resources".

```
107
108         int resourcesSize = 52000; //a few bytes more to eliminate compression fluctuation
109
```

Figure 14: resourcesSize variable

6. If Step 5 check is true then adjust "resourcesSize" variable and go to Step 1 else continue.
7. Now is virus in its first form done and it is needed to do first infection. This is performed by "FirstVirusInfector" application that is available as attachment of this work.
8. Place "virus.exe" from virus solution to "FirstVirusInfector" application.
9. Place any executable application that will server as host to "FirstVirusInfector" application and named it "victim.exe".
10. Launch "FirstVirusInfector" application.
11. Done. Infected host application will be named "infected-victim.exe". This application is fully functional and it is infected by the virus. Be careful to not launch this application in an uncontrolled environment.

Be aware that changes in "Program.cs" in virus solution will be available only for the first virus iteration. If changes are needed in other iterations of virus, it is necessary to edit "Evolution/Individuals/RawIndividual.cs" "CreateMain()" method too.

4.3 Detection testing

Testing of detection was performed by two approaches. The first one was running tester application on "Windows 10 Pro N" system with installed antivirus (free or trial) from one of the following companies:

- Avast
- Eset
- Kaspersky

Only antivirus from Avast detected a threat. Detected threat is shown in Figure 15. Detected threat name was "IDP.ALEXA.51", that is malware which uses Windows temporary folder to save "EXE" files there. This is similar behavior to virus written in this master thesis because this virus use Windows temporary folder for compilation of newly generated virus (until it cleans up).

The second approach was upload host application infected by virus (PuTTY as host application in this case) to virustotal.com. On virustotal detected this file 20 engines out of 71. Report can be found on this address `https://www.virustotal.com/#/file/c018dd72515c6207e05036918903547be4ff084b123be09ee26c278e9c751655/detection` and its part is shown in Figure 16.

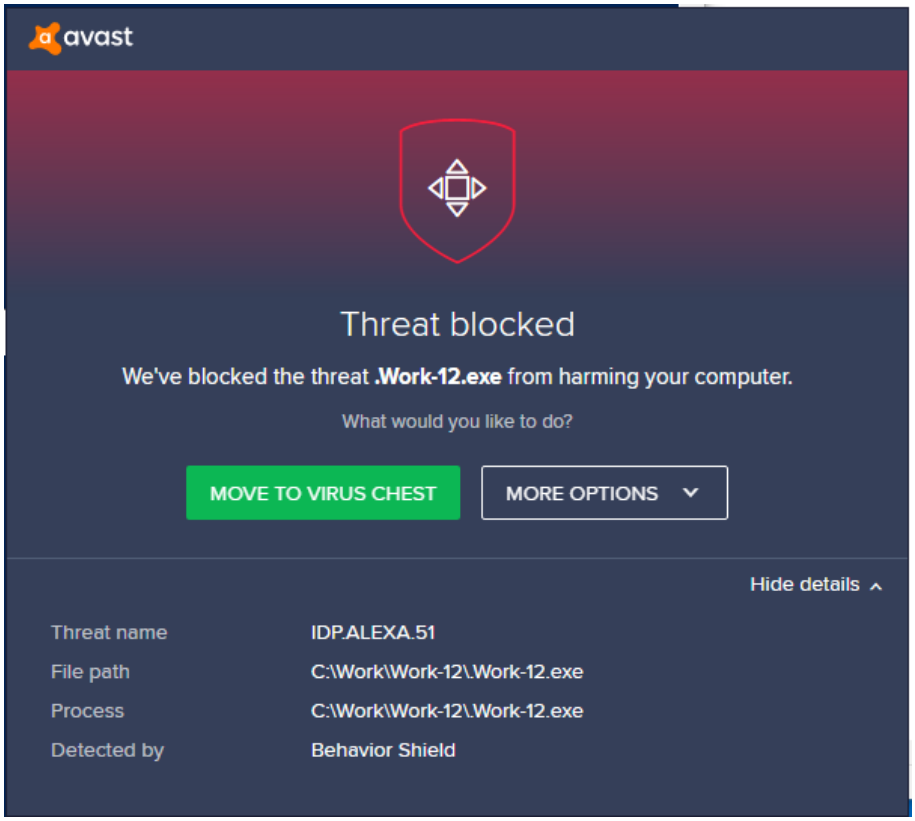

Figure 15: Avast threat detection

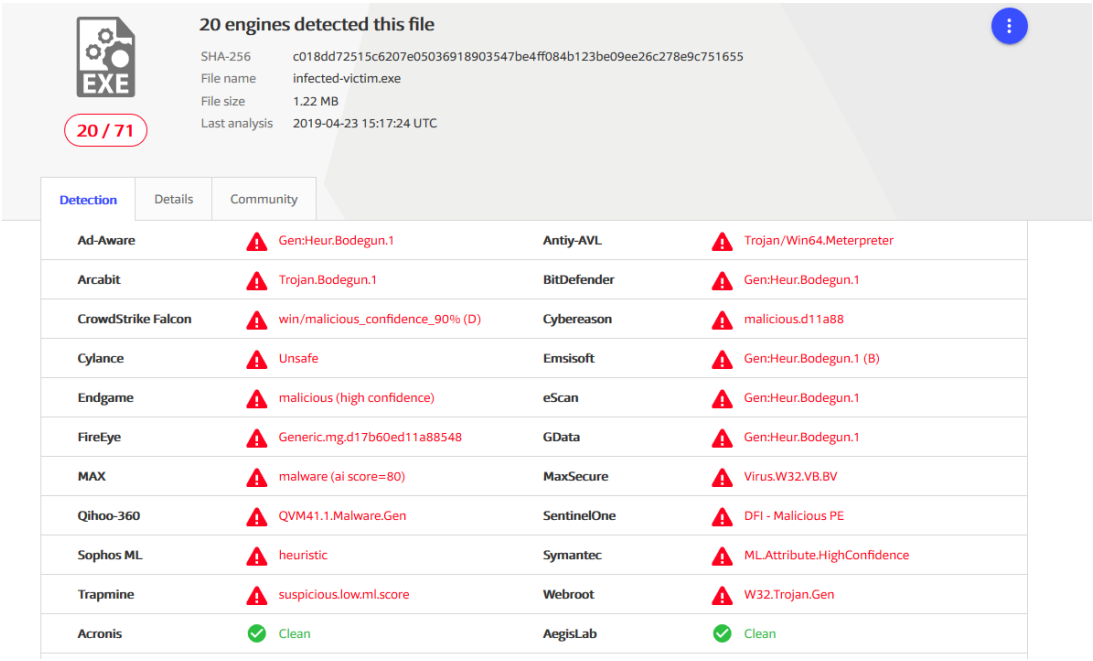

Figure 16: Virustotal detection

5 Experiment

Experiment section is divided to 3 parts - the testing environment, the course of the test and the test results.

5.1 Testing environment

Testing OS was freshly installed "Windows 10 Pro N" into Oracle VM Virtualbox. Windows 10 was then updated to latest version 1809 with OS build 17763.379. Then was created following hierarchy on file system with X replaced by the number of application (1, 2, 3 etc.):

- 30x Program - C:\Other Programs\Program-X\Program-X.exe
- 20x Game - C:\Games\Game-X\Game-X.exe
- 15x Work - C:\Work\Work-X\Work-X.exe
- 10x Downloads - C:\Users\User\Downloads\Download-X.exe

All these applications were launched twice in the same order as is listed above.

All applications listed above, except Download applications, have its shortcut on Desktop in their respective folder (Programs, Games, Work).

Next were added following shortcuts to Taskbar (Windows bottom bar):

- Program-1, Program-2, Program-3
- Game-1
- Work-1, Work-2

and to Start menu following shortcuts:

- Program-4, Program-5
- Game-2, Game-3
- Work-3, Work-4, Work-5

As the last was application "tester" placed on Desktop. This application was created to automate the testing process and it is available as an attachment of this work. This application was also edited to contain infected mark so viruses will not try to infect it.

Then was created a snapshot of OS which is used as a starting point for every test.

5.2 Testing

Total 10 virus types were created for testing. One for every implemented evolution algorithm (DE, GA, PSO, SOMA) and one that was every time randomly choosing one of the implemented algorithm, with equal chances to be taken for all algorithms. In addition to this, all previous variants were made in 2 versions. One which set the virus number of infection to 3 and the second one set the virus number of infection to 5.

Application tester was created to help automate testing. This application has as input parameters path to virus log file (virus.txt on desktop) and parameter depth. Parameter depth is used for controlling depth of launching offspring of virus. The application then work in this way:

1. Read all lines from virus.txt and save lines that are not in the processedLines list to the unprocessedLines list.
   Line is consisted of attributes of virus separated by '>' character. These attributes are (in this order):
   - "whoAmI" - attribute that contains path to virus that write this line
   - "composition" - virus structure6 of this virus
   - "whoMadeMe" - name of evolutionary algorithm which made this virus
   - "ancComposition" - virus structure6 of ancestor of this virus
   - "iInfect" - list of paths to files that this virus infected separated by '|' character
2. Launch every item from "iInfect" list (logic bomb of launched virus can prevent execution of virus logic (not host application - that is launched always))
3. Save lines from the unprocessedLines list to the processedLines list.
4. Clear the unprocessedLines list.
5. Increment actual depth variable.
6. If the actual depth is lesser than depth from the input parameter go to step 1.

The following scenario was then running for every virus type 5 times:

1. Launch script that periodically kills instances of the PuTTY application (infected host application) to avoid running tens of instances of it.
2. Place actually tested virus type (infected-victim.exe) to downloads folder
3. Launch infected-victim.exe and check if the virus was launched (logic bomb can prevent this). If not repeat this step.

4. Launch tester application with the path to virus.txt log file as parameter and parameter depth set to 5.

5. Wait for the end of the tester application

6. Save virus.txt log file and virus-exceptions.txt (if any) log file outside of the virtual machine.

7. Restore snapshot.

5.3 Experiment results

Data from all results from testing are available as an attachment of this work.

This section is focused on representation and visualization of gathered data specifically on these 5 sub-sections:

- Most used virus modules by evolution
- Average number of infection and how many times were infected all testing files
- Viruses that were created 3 or more times by evolution
- Viruses that infects 7 or more files by evolution
- Infection spreading

All previously listed sub-sections are represented and visualized separately for viruses that infects 3 victim files and for viruses that infects 5 victim files.

5.3.1 Viruses that inflict 3 victims

5.3.1.1 Most used virus modules

In Table 7 are shown most used modules that were used by 3-file infection viruses generated by random evolution. Specific modules types can be seen in bar plots - Figures 17,18,19.

Table 7: Most used virus modules by random evolution and 3-file infection

LogicBombs		**VictimSearchers**		**Infectors**		**Payloads**	
Magician2LogicBomb	35	DesktopSearcher	18	PrependInfector	107	CheersPayload	22
MagicianLogicBomb	17	StartMenuSearcher	11			CookiePayload	25
ObjLength2LogicBomb	6	TaskBarSearcher	9			GoodXYPayload	19
ObjLengthLogicBomb	20	TreeSearcher	18			IGotYouPayload	16
RandomLogicBomb	10	UserAssistSearcher	51			YRInfectedPayload	25
Time1LogicBomb	12						
Time2LogicBomb	7						

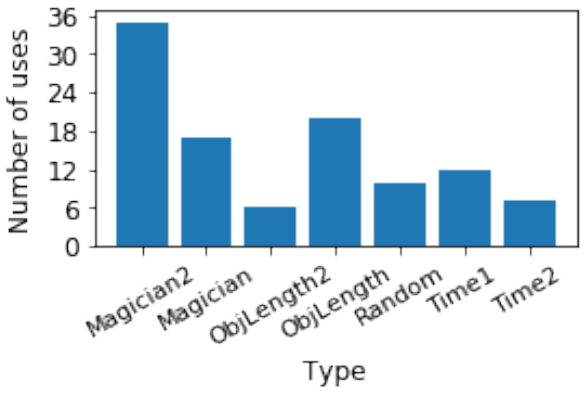

Figure 17: Number of uses for each LogicBomb type by random evolution and 3-file infection

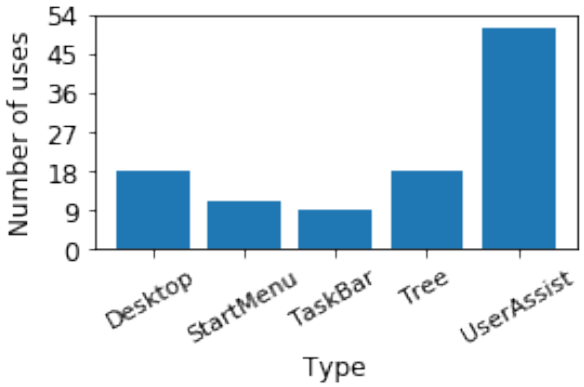

Figure 18: Number of uses for each VictimSearcher type by random evolution and 3-file infection

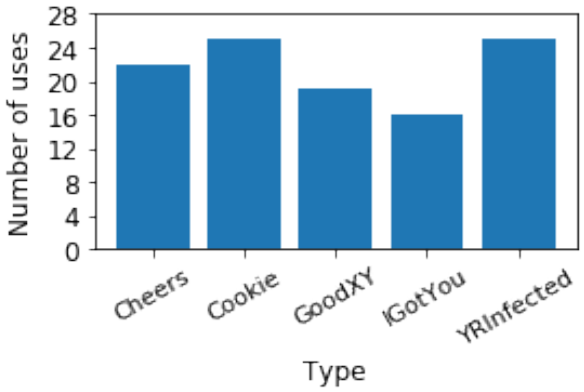

Figure 19: Number of uses for each Payload type by random evolution and 3-file infection

In Table 8 are shown most used modules that were used by 3-file infection viruses generated by DE. Specific modules types can be seen in bar plots - Figures: 20,21,22.

Table 8: Most used virus modules by DE and 3-file infection

LogicBombs		**VictimSearchers**		**Infectors**		**Payloads**	
Magician2LogicBomb	27	DesktopSearcher	24	PrependInfector	109	CheersPayload	13
MagicianLogicBomb	18	StartMenuSearcher	19			CookiePayload	28
ObjLength2LogicBomb	13	TaskBarSearcher	14			GoodXYPayload	30
ObjLengthLogicBomb	17	TreeSearcher	25			IGotYouPayload	19
RandomLogicBomb	12	UserAssistSearcher	27			YRInfectedPayload	19
Time1LogicBomb	13						
Time2LogicBomb	9						

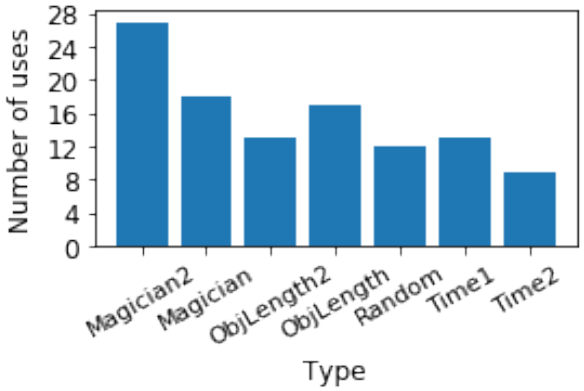

Figure 20: Number of uses for each LogicBomb type by DE and 3-file infection

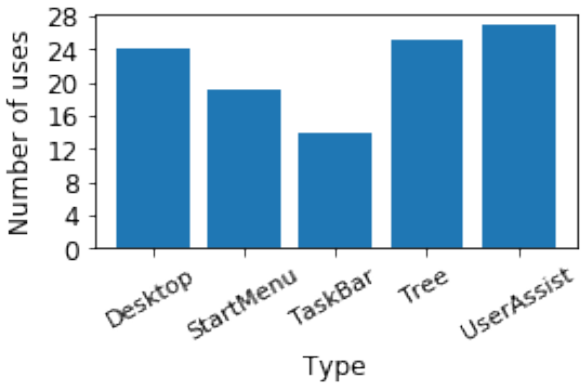

Figure 21: Number of uses for each VictimSearcher type by DE and 3-file infection

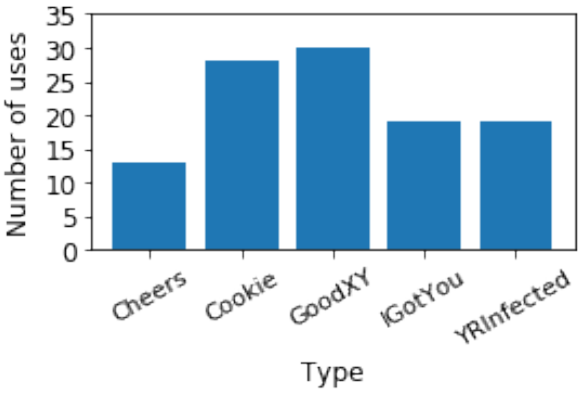

Figure 22: Number of uses for each Payload type by DE and 3-file infection

In Table 9 are shown most used modules that were used by 3-file infection viruses generated by GA. Specific modules types can be seen in bar plots - Figures: 23,24,25.

Table 9: Most used virus modules by GA and 3-file infection

LogicBombs		**VictimSearchers**		**Infectors**		**Payloads**	
Magician2LogicBomb	52	DesktopSearcher	11	PrependInfector	129	CheersPayload	30
MagicianLogicBomb	22	StartMenuSearcher	12			CookiePayload	25
ObjLength2LogicBomb	4	TaskBarSearcher	6			GoodXYPayload	28
ObjLengthLogicBomb	16	TreeSearcher	14			IGotYouPayload	23
RandomLogicBomb	7	UserAssistSearcher	86			YRInfectedPayload	23
Time1LogicBomb	22						
Time2LogicBomb	6						

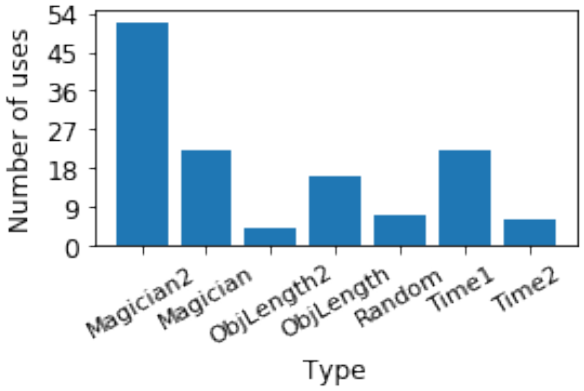

Figure 23: Number of uses for each LogicBomb type by GA and 3-file infection

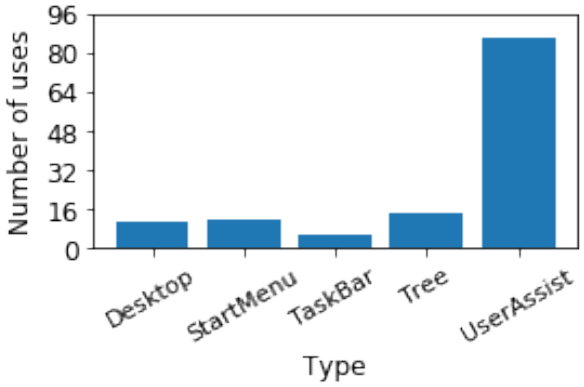

Figure 24: Number of uses for each VictimSearcher type by GA and 3-file infection

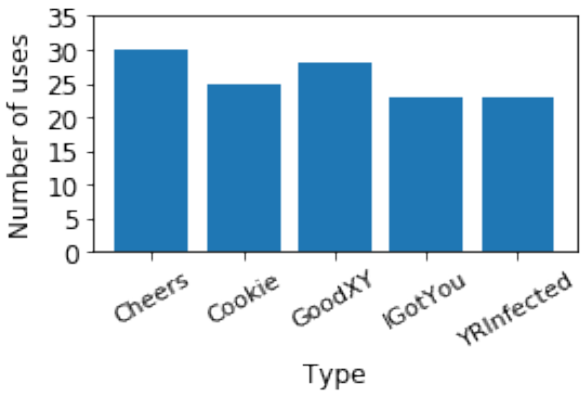

Figure 25: Number of uses for each Payload type by GA and 3-file infection

In Table 10 are shown most used modules that were used by 3-file infection viruses generated by PSO. Specific modules types can be seen in bar plots - Figures: 26,27,28.

Table 10: Most used virus modules by PSO and 3-file infection

LogicBombs		**VictimSearchers**		**Infectors**		**Payloads**	
Magician2LogicBomb	86	DesktopSearcher	7	PrependInfector	130	CheersPayload	31
MagicianLogicBomb	13	StartMenuSearcher	7			CookiePayload	24
ObjLength2LogicBomb	2	TaskBarSearcher	6			GoodXYPayload	23
ObjLengthLogicBomb	15	TreeSearcher	8			IGotYouPayload	32
RandomLogicBomb	2	UserAssistSearcher	102			YRInfectedPayload	20
Time1LogicBomb	12						
Time2LogicBomb	0						

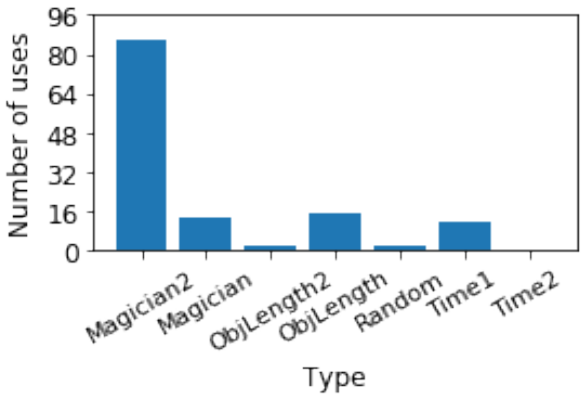

Figure 26: Number of uses for each LogicBomb type by PSO and 3-file infection

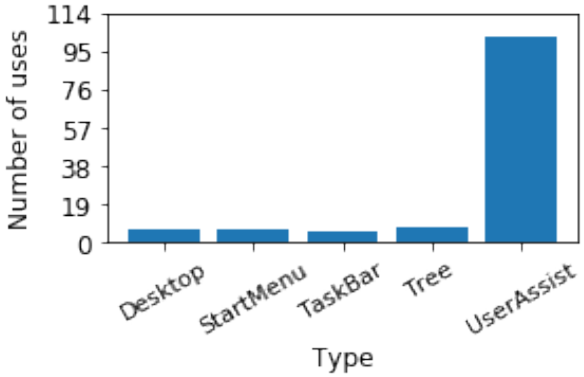

Figure 27: Number of uses for each VictimSearcher type by PSO and 3-file infection

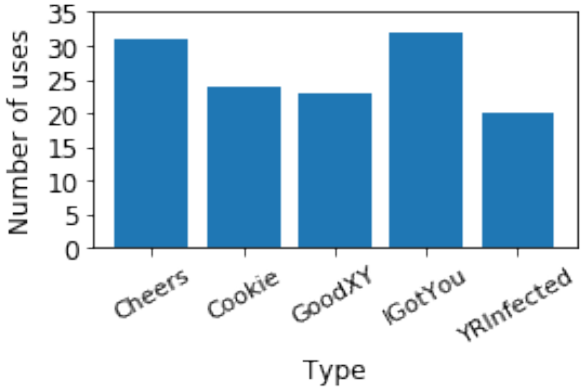

Figure 28: Number of uses for each Payload type by PSO and 3-file infection

In Table 11 are shown most used modules that were used by 3-file infection viruses generated by SOMA. Specific modules types can be seen in bar plots - Figures: 29,30,31.

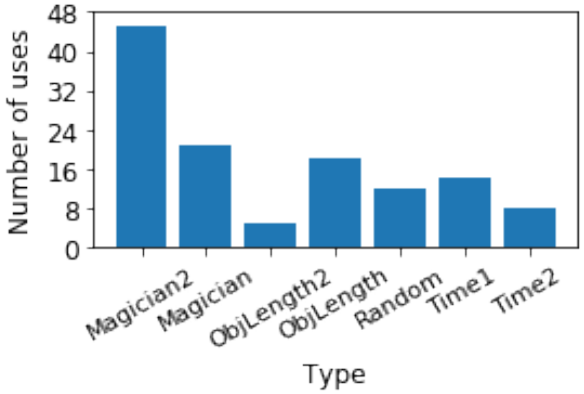

Figure 29: Number of uses for each LogicBomb type by SOMA and 3-file infection

Table 11: Most used virus modules by SOMA and 3-file infection

LogicBombs		**VictimSearchers**		**Infectors**		**Payloads**	
Magician2LogicBomb	45	DesktopSearcher	13	PrependInfector	123	CheersPayload	22
MagicianLogicBomb	21	StartMenuSearcher	15			CookiePayload	16
ObjLength2LogicBomb	5	TaskBarSearcher	5			GoodXYPayload	30
ObjLengthLogicBomb	18	TreeSearcher	20			IGotYouPayload	26
RandomLogicBomb	12	UserAssistSearcher	70			YRInfectedPayload	29
Time1LogicBomb	14						
Time2LogicBomb	8						

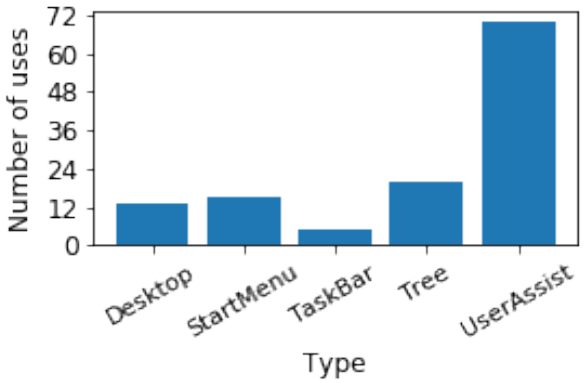

Figure 30: Number of uses for each VictimSearcher type by SOMA and 3-file infection

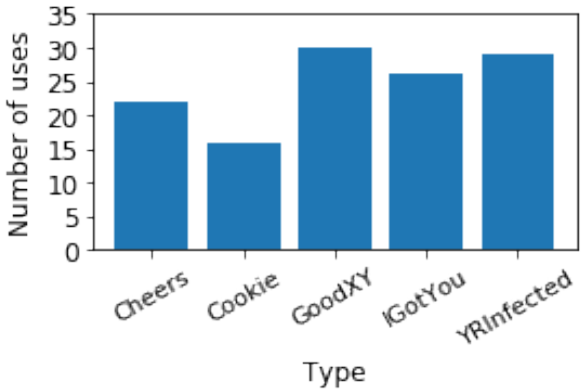

Figure 31: Number of uses for each Payload type by SOMA and 3-file infection

5.3.1.2 Average number of infection and how many times were infected all testing files

In Table 12 are showed average number of infection by evolution and how many times evolution algorithms (including random selection of evolutionary algorithm) accomplish to infect all test files.

Table 12: Average number of infection by evolution and all test file infection achievements by evolution (3-file infection viruses)

	Random	DE	GA	PSO	SOMA
Average of 5 scenarios:	61.4	62	75	75	71
Number of infecting all test files: (out of 5)	2	2	5	5	3

5.3.1.3 Viruses that were created 3 or more times

In Table 13 are shown 3-file infection viruses that were created 3 or more times by randomly selected evolution. The same can be seen in bar plot - Figure 32.

Table 13: 3-file infection viruses that were created 3 or more times by random evolution algorithm

Virus structure	Number of creations
Magician2LogicBomb,UserAssistSearcher,PrependInfector,CheersPayload(0,4,0,0)	7
Magician2LogicBomb,UserAssistSearcher,PrependInfector,YRInfectedPayload(0,4,0,4)	6
Magician2LogicBomb,UserAssistSearcher,PrependInfector,GoodXYPayload(0,4,0,2)	4
MagicianLogicBomb,UserAssistSearcher,PrependInfector,CheersPayload(1,4,0,0)	4
Magician2LogicBomb,DesktopSearcher,PrependInfector,CookiePayload(0,0,0,1)	3
ObjLengthLogicBomb,UserAssistSearcher,PrependInfector,CookiePayload(3,4,0,1)	3
Magician2LogicBomb,DesktopSearcher,PrependInfector,GoodXYPayload(0,0,0,2)	3
Magician2LogicBomb,UserAssistSearcher,PrependInfector,CookiePayload(0,4,0,1)	3

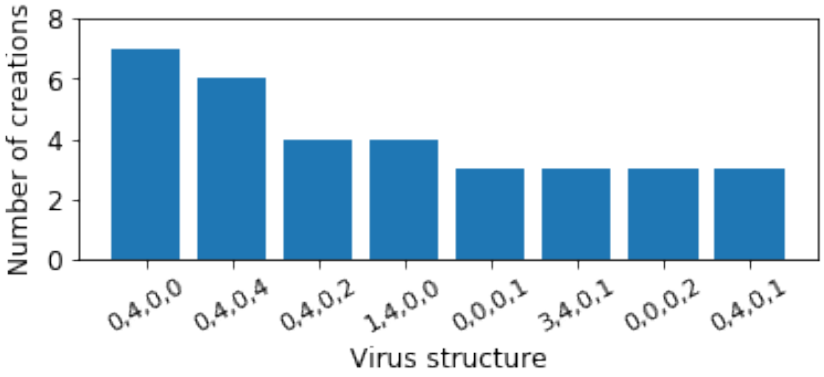

Figure 32: 3-file infection viruses that were created 3 or more times by random evolution algorithm

In Table 14 are shown 3-file infection viruses that were created 3 or more times by DE. The same can be seen in bar plot - Figure 33.

Table 14: 3-file infection viruses that were created 3 or more times by DE

Virus structure	**Number of creations**
MagicianLogicBomb,DesktopSearcher,PrependInfector,GoodXYPayload(1,0,0,2)	4
Magician2LogicBomb,TreeSearcher,PrependInfector,CookiePayload(0,3,0,1)	4
Time1LogicBomb,TreeSearcher,PrependInfector,GoodXYPayload(5,3,0,2)	3
ObjLengthLogicBomb,DesktopSearcher,PrependInfector,YRInfectedPayload(3,0,0,4)	3
Magician2LogicBomb,DesktopSearcher,PrependInfector,CookiePayload(0,0,0,1)	3
ObjLength2LogicBomb,UserAssistSearcher,PrependInfector,IGotYouPayload(2,4,0,3)	3
Magician2LogicBomb,TaskBarSearcher,PrependInfector,YRInfectedPayload(0,2,0,4)	3
MagicianLogicBomb,DesktopSearcher,PrependInfector,IGotYouPayload(1,0,0,3)	3

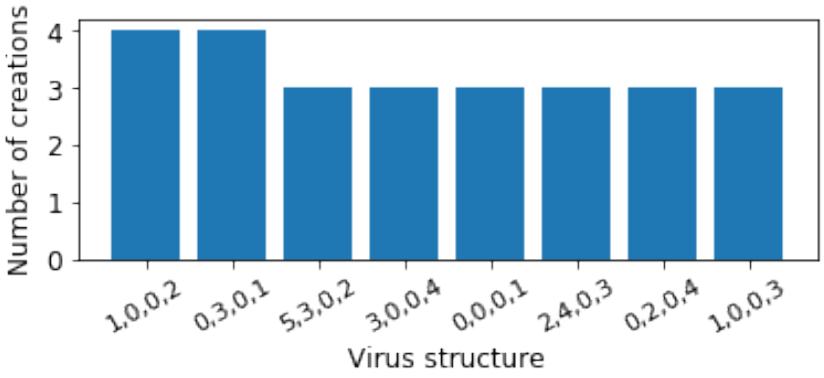

Figure 33: 3-file infection viruses that were created 3 or more times by DE

In Table 15 are shown 3-file infection viruses that were created 3 or more times by GA. The same can be seen in bar plot - Figure 34.

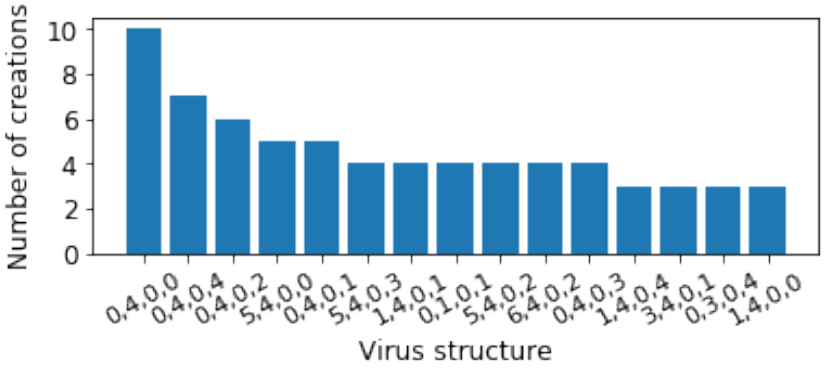

Figure 34: 3-file infection viruses that were created 3 or more times by GA

Table 15: 3-file infection viruses that were created 3 or more times by GA

Virus structure	**Number of creations**
Magician2LogicBomb,UserAssistSearcher,PrependInfector,CheersPayload(0,4,0,0)	10
Magician2LogicBomb,UserAssistSearcher,PrependInfector,YRInfectedPayload(0,4,0,4)	7
Magician2LogicBomb,UserAssistSearcher,PrependInfector,GoodXYPayload(0,4,0,2)	6
Time1LogicBomb,UserAssistSearcher,PrependInfector,CheersPayload(5,4,0,0)	5
Magician2LogicBomb,UserAssistSearcher,PrependInfector,CookiePayload(0,4,0,1)	5
Time1LogicBomb,UserAssistSearcher,PrependInfector,IGotYouPayload(5,4,0,3)	4
MagicianLogicBomb,UserAssistSearcher,PrependInfector,CookiePayload(1,4,0,1)	4
Magician2LogicBomb,StartMenuSearcher,PrependInfector,CookiePayload(0,1,0,1)	4
Time1LogicBomb,UserAssistSearcher,PrependInfector,GoodXYPayload(5,4,0,2)	4
Time2LogicBomb,UserAssistSearcher,PrependInfector,GoodXYPayload(6,4,0,2)	4
Magician2LogicBomb,UserAssistSearcher,PrependInfector,IGotYouPayload(0,4,0,3)	4
MagicianLogicBomb,UserAssistSearcher,PrependInfector,YRInfectedPayload(1,4,0,4)	3
ObjLengthLogicBomb,UserAssistSearcher,PrependInfector,CookiePayload(3,4,0,1)	3
Magician2LogicBomb,TreeSearcher,PrependInfector,YRInfectedPayload(0,3,0,4)	3
MagicianLogicBomb,UserAssistSearcher,PrependInfector,CheersPayload(1,4,0,0)	3

In Table 16 are shown 3-file infection viruses that were created 3 or more times by PSO. The same can be seen in bar plot - Figure 35.

Table 16: 3-file infection viruses that were created 3 or more times by PSO

Virus structure	**Number of creations**
Magician2LogicBomb,UserAssistSearcher,PrependInfector,IGotYouPayload(0,4,0,3)	18
Magician2LogicBomb,UserAssistSearcher,PrependInfector,YRInfectedPayload(0,4,0,4)	15
Magician2LogicBomb,UserAssistSearcher,PrependInfector,CheersPayload(0,4,0,0)	14
Magician2LogicBomb,UserAssistSearcher,PrependInfector,CookiePayload(0,4,0,1)	11
Magician2LogicBomb,UserAssistSearcher,PrependInfector,GoodXYPayload(0,4,0,2)	6
ObjLengthLogicBomb,UserAssistSearcher,PrependInfector,GoodXYPayload(3,4,0,2)	5
ObjLengthLogicBomb,UserAssistSearcher,PrependInfector,IGotYouPayload(3,4,0,3)	4
MagicianLogicBomb,UserAssistSearcher,PrependInfector,GoodXYPayload(1,4,0,2)	4
Time1LogicBomb,UserAssistSearcher,PrependInfector,CheersPayload(5,4,0,0)	4
MagicianLogicBomb,UserAssistSearcher,PrependInfector,CheersPayload(1,4,0,0)	3
Time1LogicBomb,UserAssistSearcher,PrependInfector,GoodXYPayload(5,4,0,2)	3

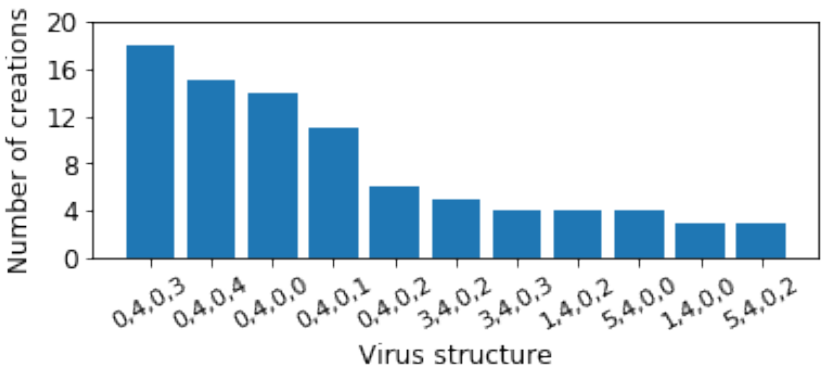

Figure 35: 3-file infection viruses that were created 3 or more times by PSO

In Table 17 are shown 3-file infection viruses that were created 3 or more times by SOMA. The same can be seen in bar plot - Figure 36.

Table 17: 3-file infection viruses that were created 3 or more times by SOMA

Virus structure	**Number of creations**
Magician2LogicBomb,UserAssistSearcher,PrependInfector,IGotYouPayload(0,4,0,3)	6
Magician2LogicBomb,UserAssistSearcher,PrependInfector,CookiePayload(0,4,0,1)	5
Magician2LogicBomb,UserAssistSearcher,PrependInfector,CheersPayload(0,4,0,0)	5
Magician2LogicBomb,UserAssistSearcher,PrependInfector,YRInfectedPayload(0,4,0,4)	5
Time1LogicBomb,UserAssistSearcher,PrependInfector,GoodXYPayload(5,4,0,2)	5
MagicianLogicBomb,UserAssistSearcher,PrependInfector,YRInfectedPayload(1,4,0,4)	4
MagicianLogicBomb,UserAssistSearcher,PrependInfector,CookiePayload(1,4,0,1)	4
ObjLengthLogicBomb,UserAssistSearcher,PrependInfector,CookiePayload(3,4,0,1)	3
MagicianLogicBomb,UserAssistSearcher,PrependInfector,GoodXYPayload(1,4,0,2)	3
ObjLengthLogicBomb,UserAssistSearcher,PrependInfector,IGotYouPayload(3,4,0,3)	3
RandomLogicBomb,UserAssistSearcher,PrependInfector,YRInfectedPayload(4,4,0,4)	3
RandomLogicBomb,UserAssistSearcher,PrependInfector,CheersPayload(4,4,0,0)	3
Magician2LogicBomb,StartMenuSearcher,PrependInfector,YRInfectedPayload(0,1,0,4)	3
Magician2LogicBomb,StartMenuSearcher,PrependInfector,GoodXYPayload(0,1,0,2)	3

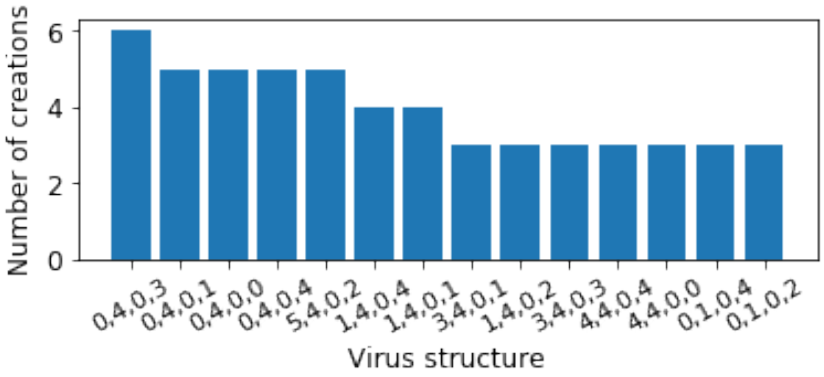

Figure 36: 3-file infection viruses that were created 3 or more times by SOMA

5.3.1.4 Viruses that infects 7 or more files

In Table 18 are shown 3-file infection viruses that infected 7 or more files by randomly selected evolution. The same can be seen in bar plot - Figure 37.

Table 18: 3-file infection viruses that infects 7 or more files by random evolutionary algorithm

Virus structure	Number of infections
Magician2LogicBomb,UserAssistSearcher,PrependInfector,CheersPayload(0,4,0,0)	21
Magician2LogicBomb,UserAssistSearcher,PrependInfector,YRInfectedPayload(0,4,0,4)	16
Magician2LogicBomb,UserAssistSearcher,PrependInfector,GoodXYPayload(0,4,0,2)	12
MagicianLogicBomb,UserAssistSearcher,PrependInfector,CheersPayload(1,4,0,0)	12
Magician2LogicBomb,DesktopSearcher,PrependInfector,CookiePayload(0,0,0,1)	9
ObjLengthLogicBomb,UserAssistSearcher,PrependInfector,CookiePayload(3,4,0,1)	9
Magician2LogicBomb,DesktopSearcher,PrependInfector,GoodXYPayload(0,0,0,2)	8
Magician2LogicBomb,UserAssistSearcher,PrependInfector,CookiePayload(0,4,0,1)	7

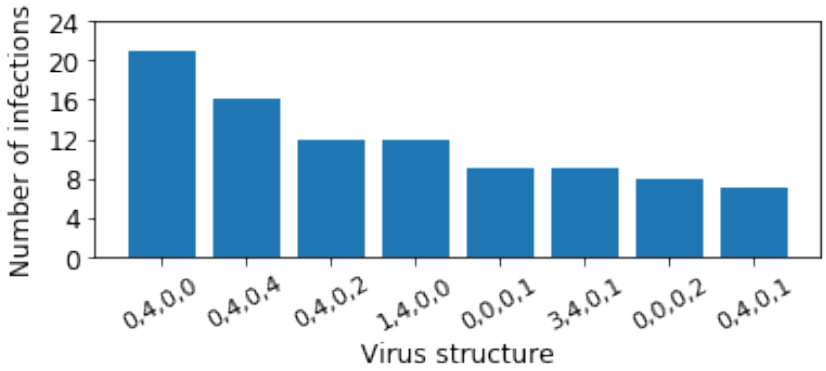

Figure 37: 3-file infection viruses that infects 7 or more files by random evolutionary algorithm

In Table 19 are shown 3-file infection viruses that infected 7 or more files by DE. The same can be seen in bar plot - Figure 38.

Table 19: 3-file infection viruses that infects 7 or more files by DE

Virus structure	Number of infections
MagicianLogicBomb,DesktopSearcher,PrependInfector,GoodXYPayload(1,0,0,2)	11
Magician2LogicBomb,TreeSearcher,PrependInfector,CookiePayload(0,3,0,1)	11
Time1LogicBomb,TreeSearcher,PrependInfector,GoodXYPayload(5,3,0,2)	9
Magician2LogicBomb,TaskBarSearcher,PrependInfector,YRInfectedPayload(0,2,0,4)	9
MagicianLogicBomb,DesktopSearcher,PrependInfector,IGotYouPayload(1,0,0,3)	9
ObjLengthLogicBomb,DesktopSearcher,PrependInfector,YRInfectedPayload(3,0,0,4)	8
Magician2LogicBomb,DesktopSearcher,PrependInfector,CookiePayload(0,0,0,1)	7
ObjLength2LogicBomb,UserAssistSearcher,PrependInfector,IGotYouPayload(2,4,0,3)	7

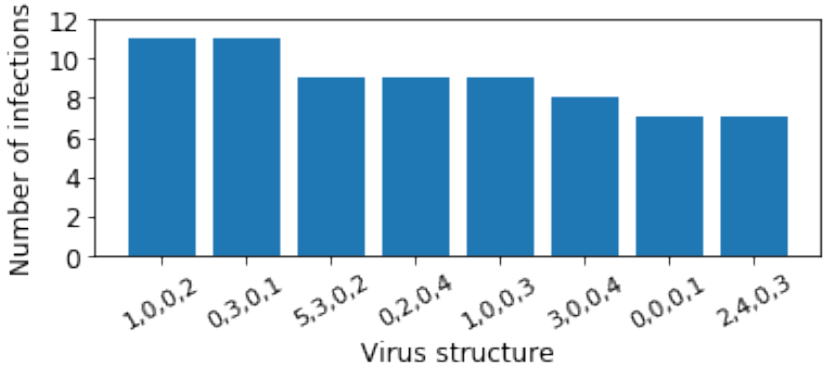

Figure 38: 3-file infection viruses that infects 7 or more files by DE

In Table 20 are shown 3-file infection viruses that infected 7 or more files by GA. The same can be seen in bar plot - Figure 39.

Table 20: 3-file infection viruses that infects 7 or more files by GA

Virus structure	**Number of infections**
Magician2LogicBomb,UserAssistSearcher,PrependInfector,CheersPayload(0,4,0,0)	30
Magician2LogicBomb,UserAssistSearcher,PrependInfector,YRInfectedPayload(0,4,0,4)	21
Magician2LogicBomb,UserAssistSearcher,PrependInfector,GoodXYPayload(0,4,0,2)	18
Time1LogicBomb,UserAssistSearcher,PrependInfector,CheersPayload(5,4,0,0)	15
Magician2LogicBomb,UserAssistSearcher,PrependInfector,CookiePayload(0,4,0,1)	14
MagicianLogicBomb,UserAssistSearcher,PrependInfector,CookiePayload(1,4,0,1)	12
Time1LogicBomb,UserAssistSearcher,PrependInfector,GoodXYPayload(5,4,0,2)	12
Time2LogicBomb,UserAssistSearcher,PrependInfector,GoodXYPayload(6,4,0,2)	12
Magician2LogicBomb,UserAssistSearcher,PrependInfector,IGotYouPayload(0,4,0,3)	12
Time1LogicBomb,UserAssistSearcher,PrependInfector,IGotYouPayload(5,4,0,3)	11
Magician2LogicBomb,StartMenuSearcher,PrependInfector,CookiePayload(0,1,0,1)	10
MagicianLogicBomb,UserAssistSearcher,PrependInfector,YRInfectedPayload(1,4,0,4)	9
ObjLengthLogicBomb,UserAssistSearcher,PrependInfector,CookiePayload(3,4,0,1)	9
Magician2LogicBomb,TreeSearcher,PrependInfector,YRInfectedPayload(0,3,0,4)	9
MagicianLogicBomb,UserAssistSearcher,PrependInfector,CheersPayload(1,4,0,0)	9

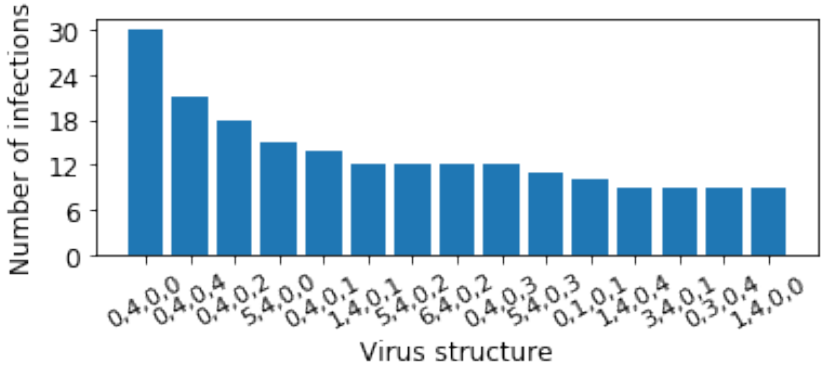

Figure 39: 3-file infection viruses that infects 7 or more files by GA

In Table 21 are shown 3-file infection viruses that infected 7 or more files by PSO. The same can be seen in bar plot - Figure 40.

Table 21: 3-file infection viruses that infects 7 or more files by PSO

Virus structure	**Number of infections**
Magician2LogicBomb,UserAssistSearcher,PrependInfector,IGotYouPayload(0,4,0,3)	53
Magician2LogicBomb,UserAssistSearcher,PrependInfector,YRInfectedPayload(0,4,0,4)	43
Magician2LogicBomb,UserAssistSearcher,PrependInfector,CheersPayload(0,4,0,0)	42
Magician2LogicBomb,UserAssistSearcher,PrependInfector,CookiePayload(0,4,0,1)	31
Magician2LogicBomb,UserAssistSearcher,PrependInfector,GoodXYPayload(0,4,0,2)	17
ObjLengthLogicBomb,UserAssistSearcher,PrependInfector,GoodXYPayload(3,4,0,2)	15
ObjLengthLogicBomb,UserAssistSearcher,PrependInfector,IGotYouPayload(3,4,0,3)	12
MagicianLogicBomb,UserAssistSearcher,PrependInfector,GoodXYPayload(1,4,0,2)	12
Time1LogicBomb,UserAssistSearcher,PrependInfector,CheersPayload(5,4,0,0)	10
MagicianLogicBomb,UserAssistSearcher,PrependInfector,CheersPayload(1,4,0,0)	9
Time1LogicBomb,UserAssistSearcher,PrependInfector,GoodXYPayload(5,4,0,2)	9

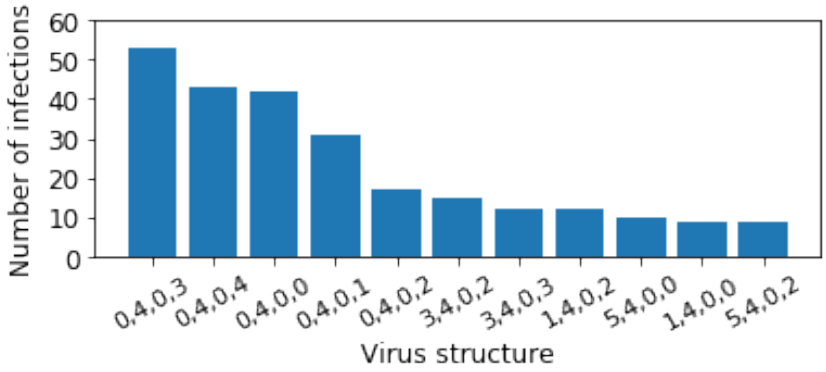

Figure 40: 3-file infection viruses that infects 7 or more files by PSO

In Table 22 are shown 3-file infection viruses that infected 7 or more files by SOMA. The same can be seen in bar plot - Figure 41.

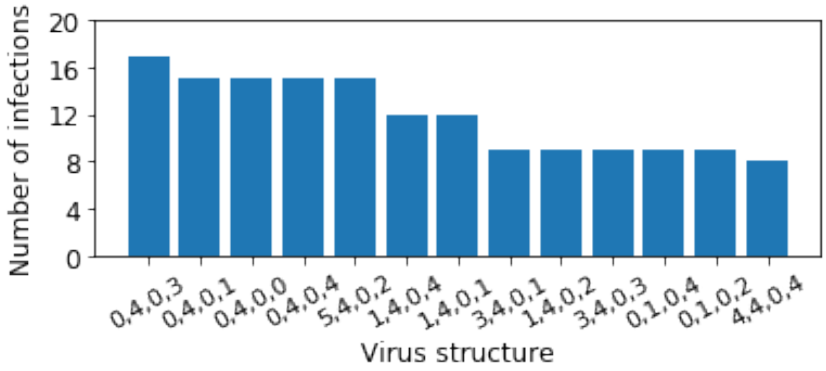

Figure 41: 3-file infection viruses that infects 7 or more files by SOMA

Table 22: 3-file infection viruses that infects 7 or more files by SOMA

Virus structure	Number of infections
Magician2LogicBomb,UserAssistSearcher,PrependInfector,IGotYouPayload(0,4,0,3)	17
Magician2LogicBomb,UserAssistSearcher,PrependInfector,CookiePayload(0,4,0,1)	15
Magician2LogicBomb,UserAssistSearcher,PrependInfector,CheersPayload(0,4,0,0)	15
Magician2LogicBomb,UserAssistSearcher,PrependInfector,YRInfectedPayload(0,4,0,4)	15
Time1LogicBomb,UserAssistSearcher,PrependInfector,GoodXYPayload(5,4,0,2)	15
MagicianLogicBomb,UserAssistSearcher,PrependInfector,YRInfectedPayload(1,4,0,4)	12
MagicianLogicBomb,UserAssistSearcher,PrependInfector,CookiePayload(1,4,0,1)	12
ObjLengthLogicBomb,UserAssistSearcher,PrependInfector,CookiePayload(3,4,0,1)	9
MagicianLogicBomb,UserAssistSearcher,PrependInfector,GoodXYPayload(1,4,0,2)	9
ObjLengthLogicBomb,UserAssistSearcher,PrependInfector,IGotYouPayload(3,4,0,3)	9
Magician2LogicBomb,StartMenuSearcher,PrependInfector,YRInfectedPayload(0,1,0,4)	9
Magician2LogicBomb,StartMenuSearcher,PrependInfector,GoodXYPayload(0,1,0,2)	9
RandomLogicBomb,UserAssistSearcher,PrependInfector,YRInfectedPayload(4,4,0,4)	8

5.3.1.5 Infection visualization

For infection visualization were generated files in Gephi (.gdf) format. All this files are available as attachment in data folder of this work. For visualization of 3-file infect virus was chosen data "data3-random-1.gdf" which means:

- data3 - data from virus that is 3-file infection
- random - evolutionary algorithm is randomly selected
- 1 - it was 1st out of 5 iterations of scenario

For visualization then were created 5 images showing testing files on file system:

- Groups with node names (Figure 42) - shows application group (Program/Game/Work/-Download) and application name
- Groups (Figure 43) - shows application group (Program/Game/Work/Download)
- Spreading heat-map (Figure 44) - shows infection progression
- VictimSearchers (Figure 45) - shows which type of VictimSearcher was used by node (virus) to infect others
- Evolution (Figure 46) - shows which evolution created node (virus)

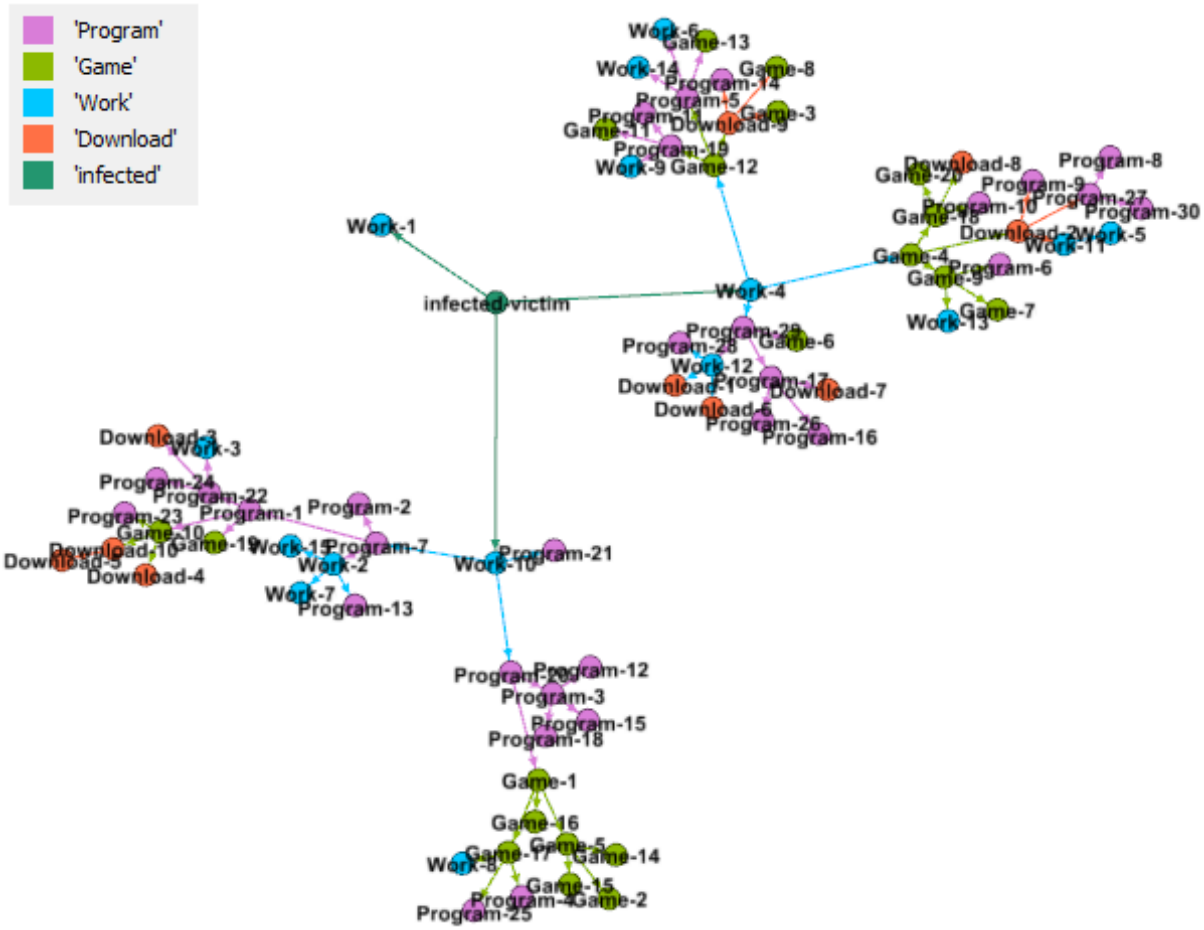

Figure 42: Application group distinguished by color and application names (data3-random-1.gdf)

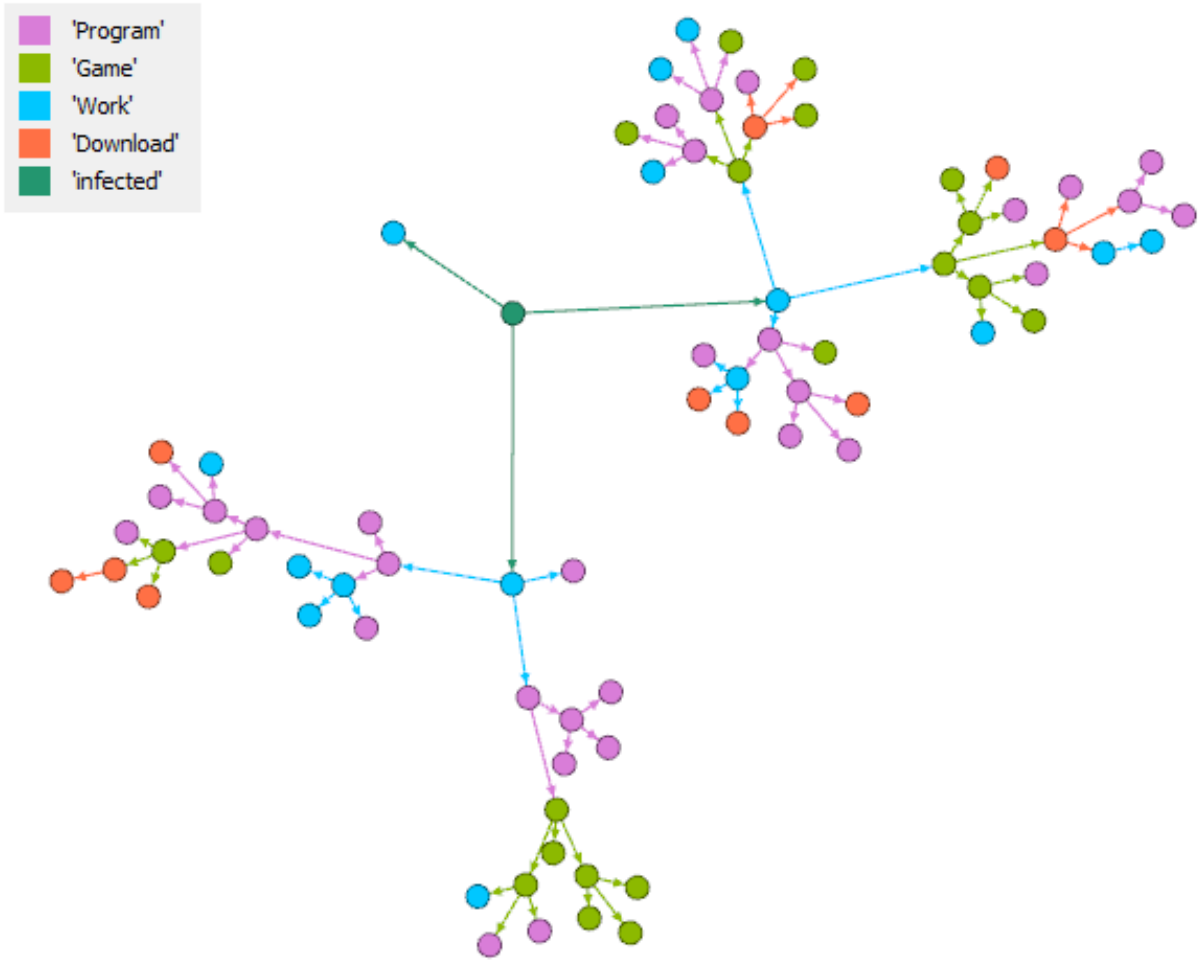

Figure 43: Application group distinguished by color (data3-random-1.gdf)

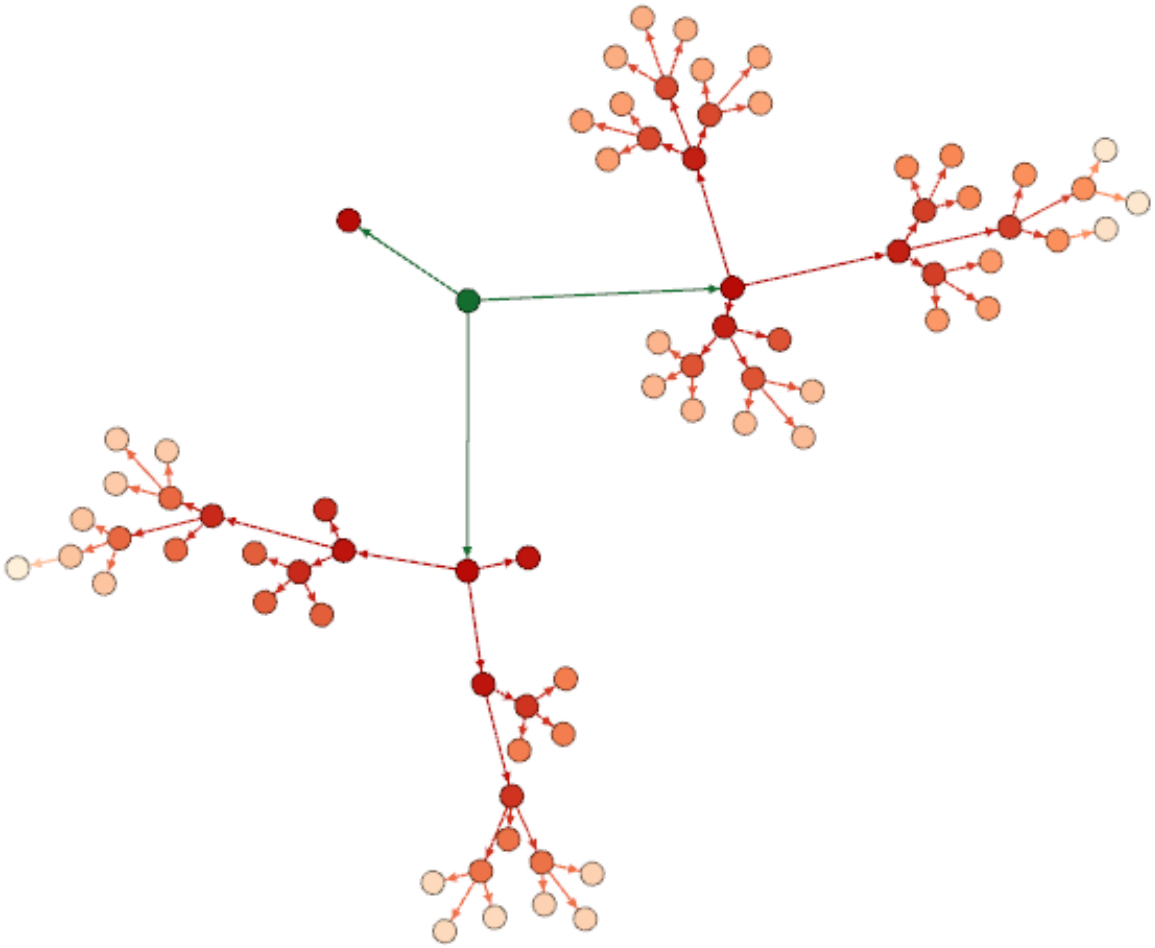

Figure 44: Heat-map of applications infections. Darker nodes were infected earlier. Green node is "infected-victim.exe" first infected file. (data3-random-1.gdf)

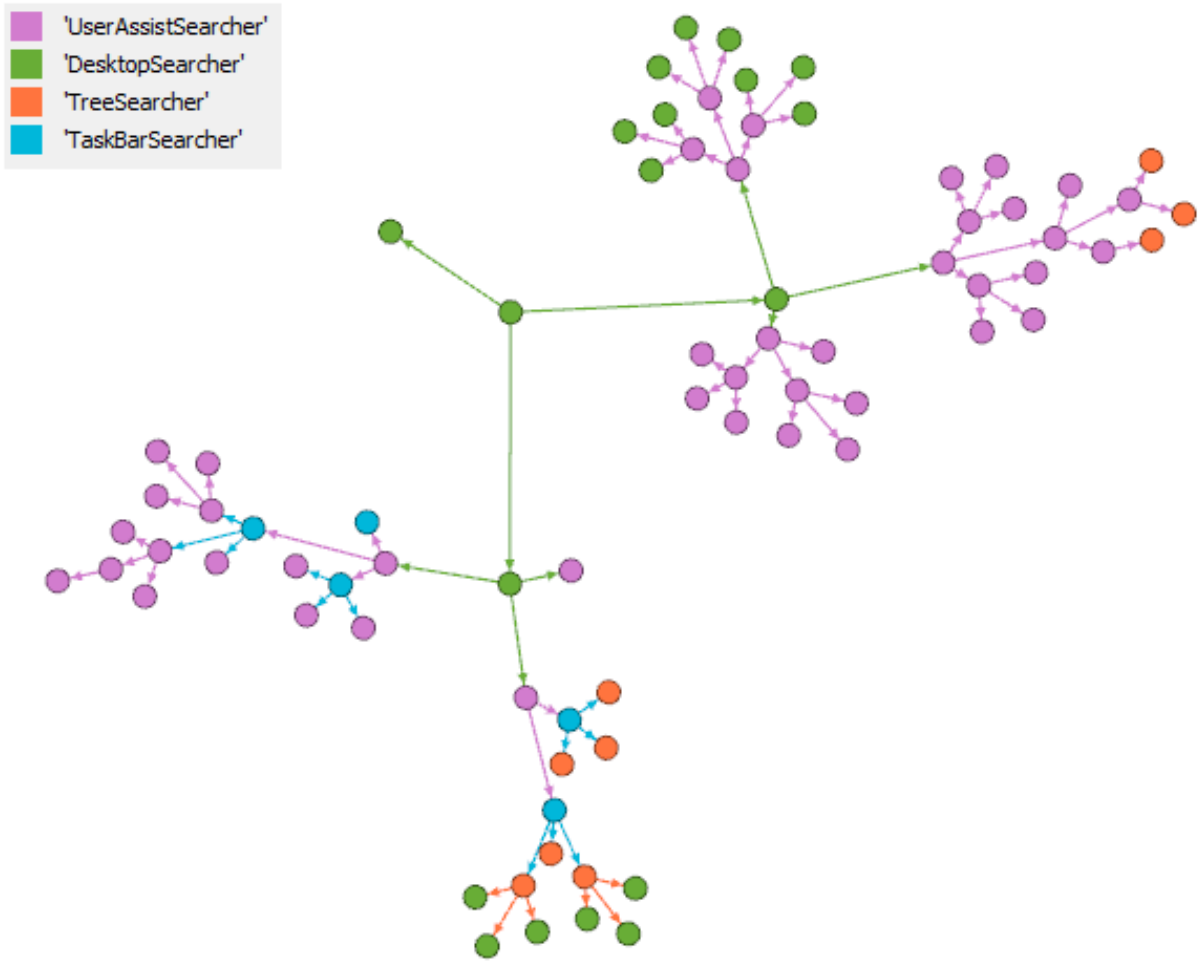

Figure 45: Node (virus) VictimSearcher distinguished by color (data3-random-1.gdf)

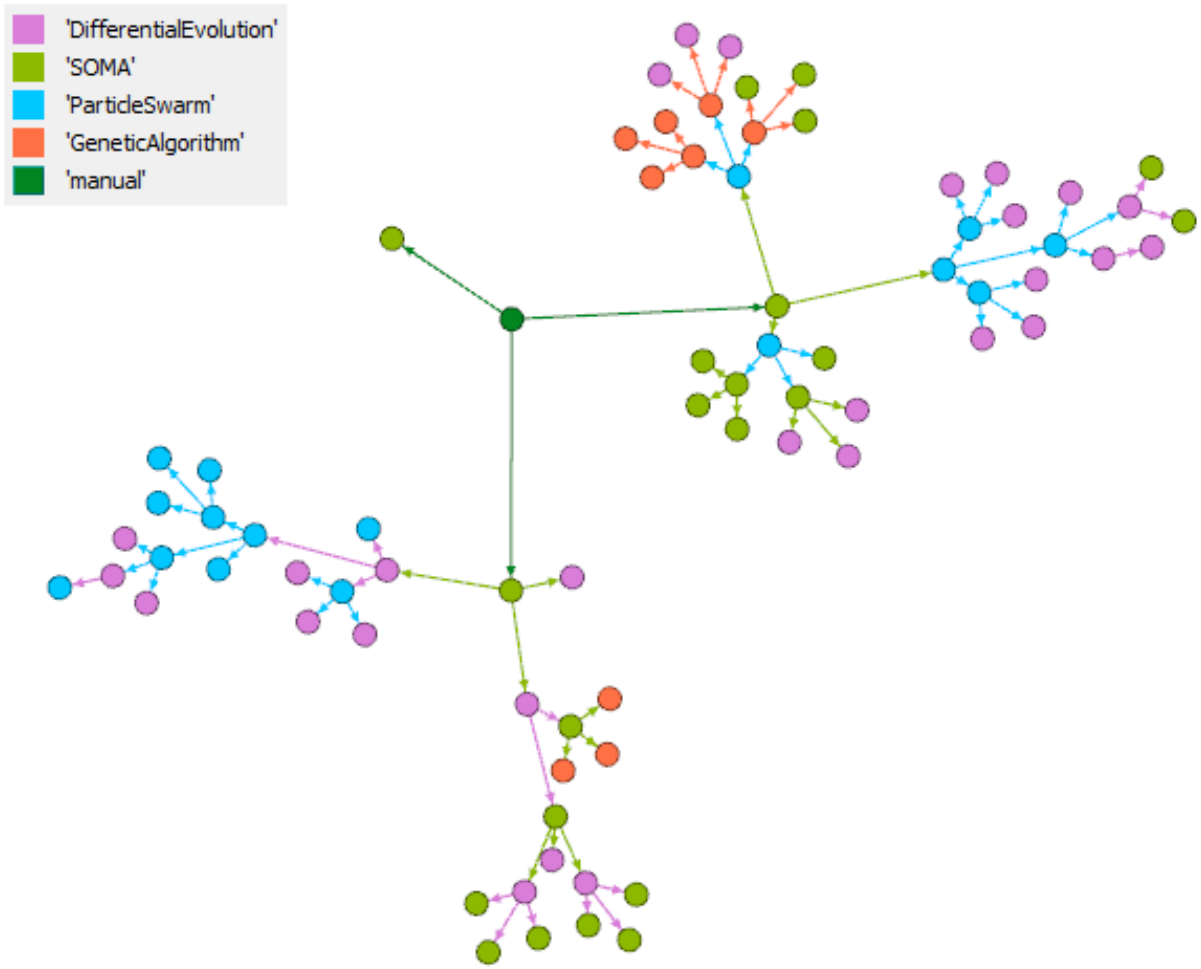

Figure 46: By which evolutionary algorithm was node (virus) created distinguished by color (data3-random-1.gdf)

5.3.2 Viruses that inflict 5 victims

5.3.2.1 Most used virus modules

In Table 23 are shown most used modules that were used by 5-file infection viruses generated by random evolution. Specific modules types can be seen in bar plots - Figures 47,48,49.

Table 23: Most used virus modules by random evolution and 5-file infection

LogicBombs		**VictimSearchers**		**Infectors**		**Payloads**	
Magician2LogicBomb	39	DesktopSearcher	6	PrependInfector	79	CheersPayload	17
MagicianLogicBomb	12	StartMenuSearcher	12			CookiePayload	20
ObjLength2LogicBomb	8	TaskBarSearcher	7			GoodXYPayload	14
ObjLengthLogicBomb	3	TreeSearcher	11			IGotYouPayload	11
RandomLogicBomb	7	UserAssistSearcher	43			YRInfectedPayload	17
Time1LogicBomb	6						
Time2LogicBomb	4						

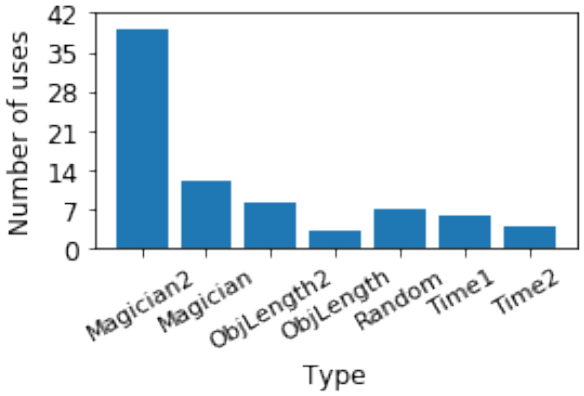

Figure 47: Number of uses for each LogicBomb type by random evolution and 5-file infection

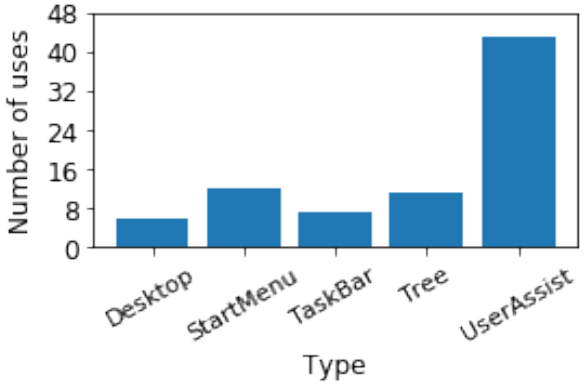

Figure 48: Number of uses for each VictimSearcher type by random evolution and 5-file infection

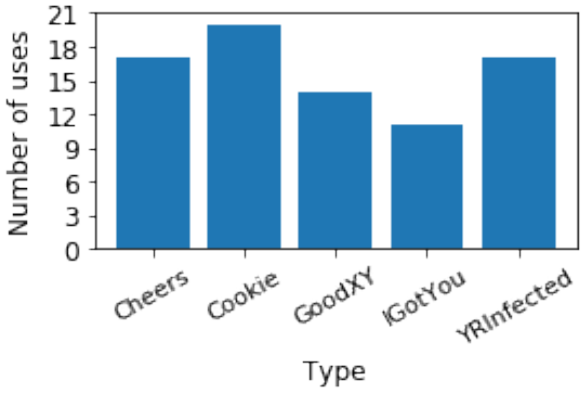

Figure 49: Number of uses for each Payload type by random evolution and 5-file infection

In Table 24 are shown most used modules that were used by 5-file infection viruses generated by DE. Specific modules types can be seen in bar plots - Figures: 50,51,52.

Table 24: Most used virus modules by DE and 5-file infection

LogicBombs		**VictimSearchers**		**Infectors**		**Payloads**	
Magician2LogicBomb	21	DesktopSearcher	13	PrependInfector	84	CheersPayload	26
MagicianLogicBomb	12	StartMenuSearcher	15			CookiePayload	18
ObjLength2LogicBomb	13	TaskBarSearcher	21			GoodXYPayload	17
ObjLengthLogicBomb	8	TreeSearcher	13			IGotYouPayload	14
RandomLogicBomb	9	UserAssistSearcher	22			YRInfectedPayload	9
Time1LogicBomb	9						
Time2LogicBomb	12						

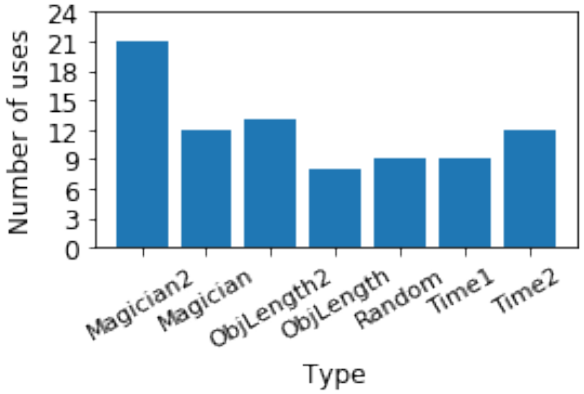

Figure 50: Number of uses for each LogicBomb type by DE and 5-file infection

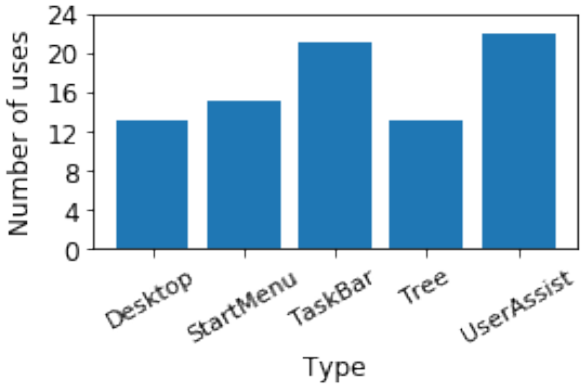

Figure 51: Number of uses for each VictimSearcher type by DE and 5-file infection

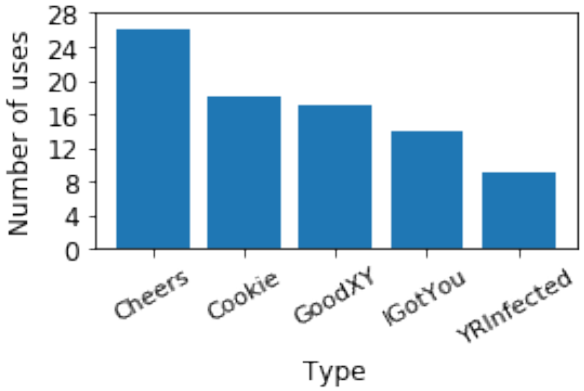

Figure 52: Number of uses for each Payload type by DE and 5-file infection

In Table 25 are shown most used modules that were used by 5-file infection viruses generated by GA. Specific modules types can be seen in bar plots - Figures: 53,54,55.

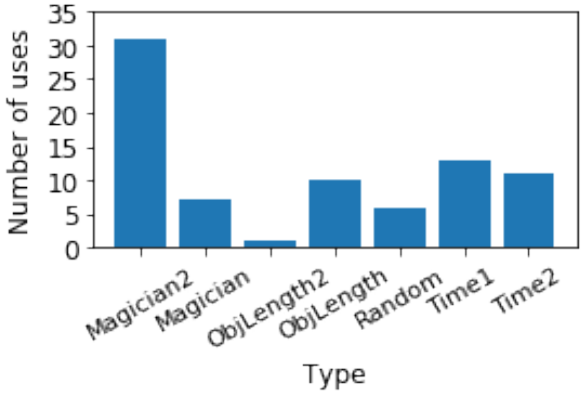

Figure 53: Number of uses for each LogicBomb type by GA and 5-file infection

Table 25: Most used virus modules by GA and 5-file infection

LogicBombs		**VictimSearchers**		**Infectors**		**Payloads**	
Magician2LogicBomb	31	DesktopSearcher	5	PrependInfector	79	CheersPayload	19
MagicianLogicBomb	7	StartMenuSearcher	6			CookiePayload	15
ObjLength2LogicBomb	1	TaskBarSearcher	7			GoodXYPayload	16
ObjLengthLogicBomb	10	TreeSearcher	7			IGotYouPayload	15
RandomLogicBomb	6	UserAssistSearcher	54			YRInfectedPayload	14
Time1LogicBomb	13						
Time2LogicBomb	11						

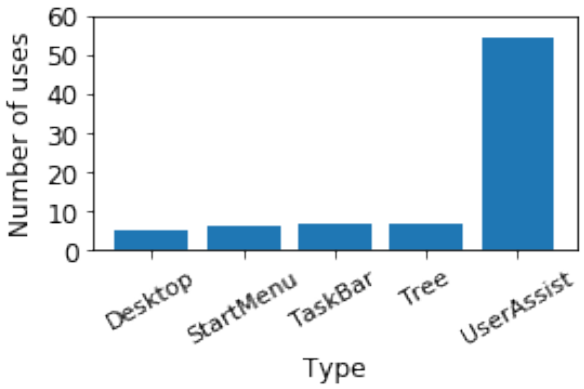

Figure 54: Number of uses for each VictimSearcher type by GA and 5-file infection

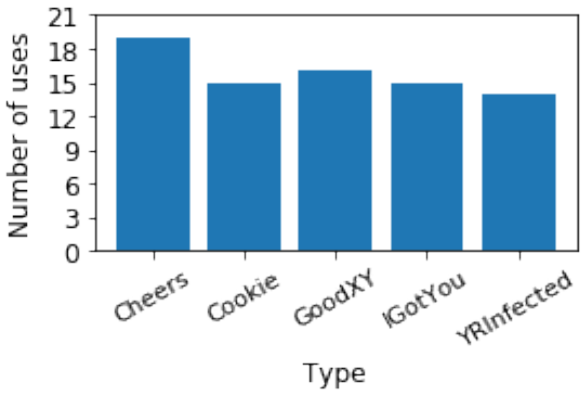

Figure 55: Number of uses for each Payload type by GA and 5-file infection

In Table 26 are shown most used modules that were used by 5-file infection viruses generated by PSO. Specific modules types can be seen in bar plots - Figures: 56,57,58.

Table 26: Most used virus modules by PSO and 5-file infection

LogicBombs		**VictimSearchers**		**Infectors**		**Payloads**	
Magician2LogicBomb	47	DesktopSearcher	1	PrependInfector	77	CheersPayload	12
MagicianLogicBomb	13	StartMenuSearcher	2			CookiePayload	17
ObjLength2LogicBomb	0	TaskBarSearcher	2			GoodXYPayload	16
ObjLengthLogicBomb	11	TreeSearcher	5			IGotYouPayload	13
RandomLogicBomb	1	UserAssistSearcher	67			YRInfectedPayload	19
Time1LogicBomb	5						
Time2LogicBomb	0						

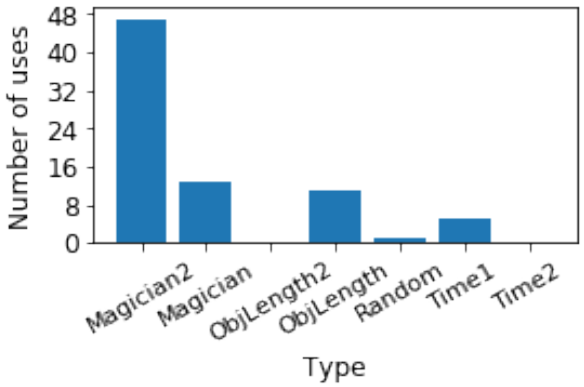

Figure 56: Number of uses for each LogicBomb type by PSO and 5-file infection

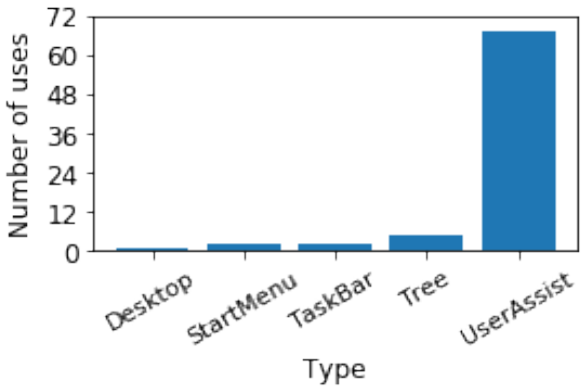

Figure 57: Number of uses for each VictimSearcher type by PSO and 5-file infection

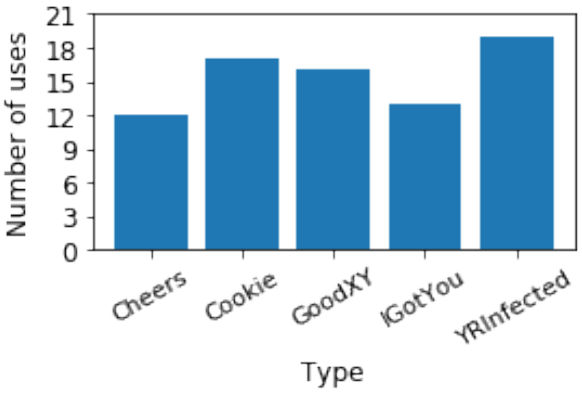

Figure 58: Number of uses for each Payload type by PSO and 5-file infection

In Table 27 are shown most used modules that were used by 5-file infection viruses generated by SOMA. Specific modules types can be seen in bar plots - Figures: 59,60,61.

Table 27: Most used virus modules by SOMA and 5-file infection

LogicBombs		**VictimSearchers**		**Infectors**		**Payloads**	
Magician2LogicBomb	31	DesktopSearcher	4	PrependInfector	78	CheersPayload	20
MagicianLogicBomb	14	StartMenuSearcher	9			CookiePayload	17
ObjLength2LogicBomb	4	TaskBarSearcher	5			GoodXYPayload	14
ObjLengthLogicBomb	7	TreeSearcher	10			IGotYouPayload	9
RandomLogicBomb	6	UserAssistSearcher	50			YRInfectedPayload	18
Time1LogicBomb	9						
Time2LogicBomb	7						

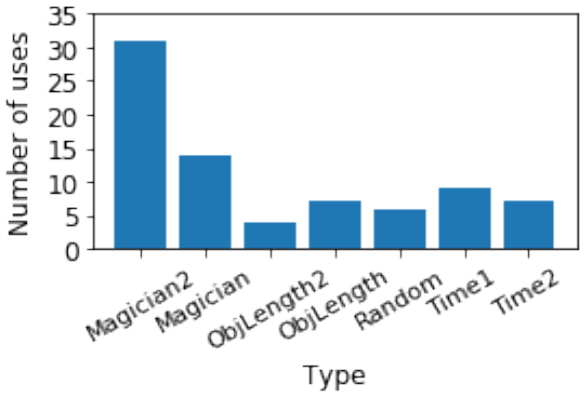

Figure 59: Number of uses for each LogicBomb type by SOMA and 5-file infection

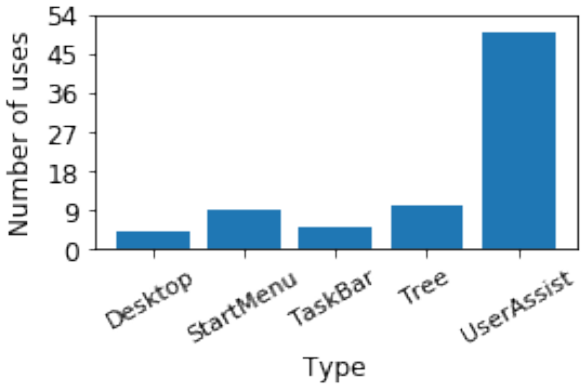

Figure 60: Number of uses for each VictimSearcher type by SOMA and 5-file infection

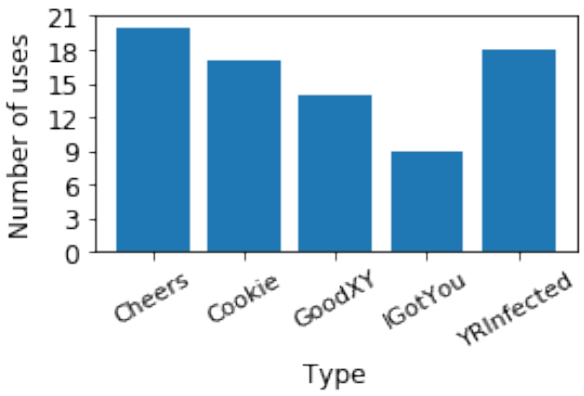

Figure 61: Number of uses for each Payload type by SOMA and 5-file infection

5.3.2.2 Average number of infection and how many times were infected all testing files

In Table 28 are showed average number of infection by evolution and how many times evolution algorithms (including random selection of evolutionary algorithm) accomplish to infect all test files.

Table 28: Average number of infection by evolution and all test file infection achievements by evolution (5-file infection viruses)

	Random	**DE**	**GA**	**PSO**	**SOMA**
Average of 5 scenarios:	75	75	75	75	75
Number of infecting all test files: (out of 5)	5	5	5	5	5

5.3.2.3 Viruses that were created 3 or more times

In Table 29 are shown 5-file infection viruses that were created 3 or more times by randomly selected evolution. The same can be seen in bar plot - Figure 62.

Table 29: 5-file infection viruses that were created 3 or more times by random evolution algorithm

Virus structure	**Number of creations**
Magician2LogicBomb,UserAssistSearcher,PrependInfector,YRInfectedPayload(0,4,0,4)	7
Magician2LogicBomb,UserAssistSearcher,PrependInfector,CheersPayload(0,4,0,0)	5
Magician2LogicBomb,UserAssistSearcher,PrependInfector,CookiePayload(0,4,0,1)	4
ObjLength2LogicBomb,UserAssistSearcher,PrependInfector,CookiePayload(2,4,0,1)	4
Magician2LogicBomb,UserAssistSearcher,PrependInfector,GoodXYPayload(0,4,0,2)	3
Magician2LogicBomb,TreeSearcher,PrependInfector,YRInfectedPayload(0,3,0,4)	3
Magician2LogicBomb,TaskBarSearcher,PrependInfector,YRInfectedPayload(0,2,0,4)	3

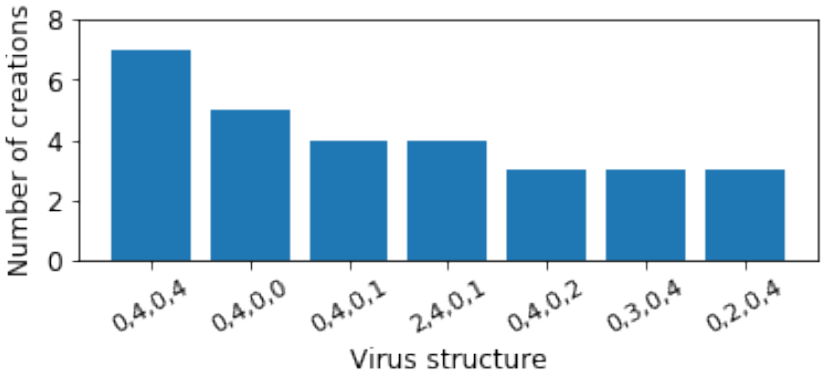

Figure 62: 5-file infection viruses that were created 3 or more times by random evolution algorithm

In Table 30 are shown 5-file infection viruses that were created 3 or more times by DE. The same can be seen in bar plot - Figure 63.

Table 30: 5-file infection viruses that were created 3 or more times by DE

Virus structure	**Number of creations**
ObjLengthLogicBomb,TaskBarSearcher,PrependInfector,CheersPayload(3,2,0,0)	4
Magician2LogicBomb,TaskBarSearcher,PrependInfector,CheersPayload(0,2,0,0)	3
Time2LogicBomb,UserAssistSearcher,PrependInfector,CookiePayload(6,4,0,1)	3

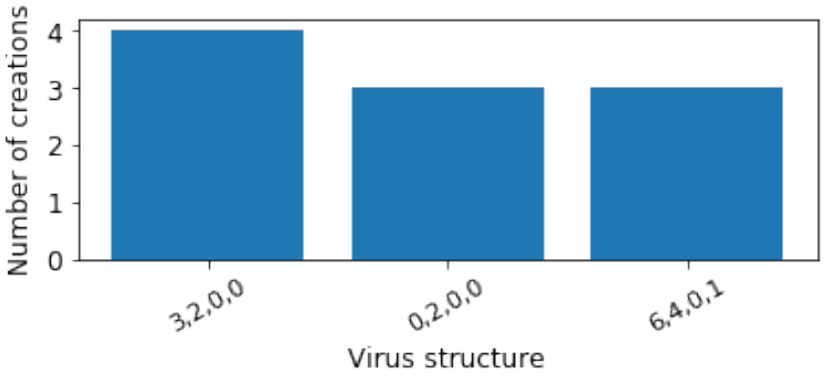

Figure 63: 5-file infection viruses that were created 3 or more times by DE

In Table 31 are shown 5-file infection viruses that were created 3 or more times by GA. The same can be seen in bar plot - Figure 64.

Table 31: 5-file infection viruses that were created 3 or more times by GA

Virus structure	Number of creations
Magician2LogicBomb,UserAssistSearcher,PrependInfector,CookiePayload(0,4,0,1)	6
Magician2LogicBomb,UserAssistSearcher,PrependInfector,CheersPayload(0,4,0,0)	5
Magician2LogicBomb,UserAssistSearcher,PrependInfector,GoodXYPayload(0,4,0,2)	3
Time2LogicBomb,UserAssistSearcher,PrependInfector,CheersPayload(6,4,0,0)	3
ObjLengthLogicBomb,UserAssistSearcher,PrependInfector,CheersPayload(3,4,0,0)	3
ObjLengthLogicBomb,UserAssistSearcher,PrependInfector,GoodXYPayload(3,4,0,2)	3
Time2LogicBomb,UserAssistSearcher,PrependInfector,YRInfectedPayload(6,4,0,4)	3

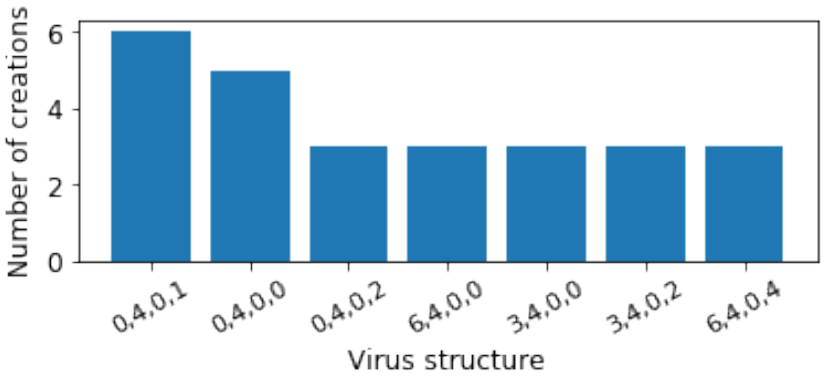

Figure 64: 5-file infection viruses that were created 3 or more times by GA

In Table 32 are shown 5-file infection viruses that were created 3 or more times by PSO. The same can be seen in bar plot - Figure 65.

Table 32: 5-file infection viruses that were created 3 or more times by PSO

Virus structure	Number of creations
Magician2LogicBomb,UserAssistSearcher,PrependInfector,YRInfectedPayload(0,4,0,4)	11
Magician2LogicBomb,UserAssistSearcher,PrependInfector,CheersPayload(0,4,0,0)	9
Magician2LogicBomb,UserAssistSearcher,PrependInfector,GoodXYPayload(0,4,0,2)	9
Magician2LogicBomb,UserAssistSearcher,PrependInfector,IGotYouPayload(0,4,0,3)	7
MagicianLogicBomb,UserAssistSearcher,PrependInfector,CookiePayload(1,4,0,1)	4
MagicianLogicBomb,UserAssistSearcher,PrependInfector,YRInfectedPayload(1,4,0,4)	4
Magician2LogicBomb,UserAssistSearcher,PrependInfector,CookiePayload(0,4,0,1)	4
ObjLengthLogicBomb,UserAssistSearcher,PrependInfector,CookiePayload(3,4,0,1)	4
MagicianLogicBomb,UserAssistSearcher,PrependInfector,IGotYouPayload(1,4,0,3)	3

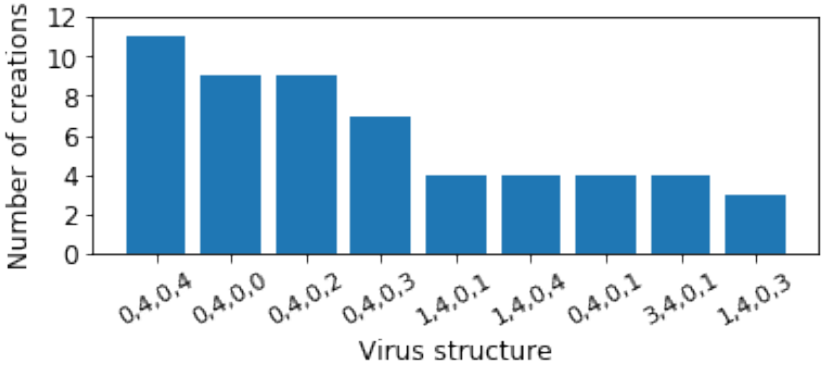

Figure 65: 5-file infection viruses that were created 3 or more times by PSO

In Table 33 are shown 5-file infection viruses that were created 3 or more times by SOMA. The same can be seen in bar plot - Figure 66.

Table 33: 5-file infection viruses that were created 3 or more times by SOMA

Virus structure	**Number of creations**
Magician2LogicBomb,UserAssistSearcher,PrependInfector,GoodXYPayload(0,4,0,2)	4
Magician2LogicBomb,UserAssistSearcher,PrependInfector,CookiePayload(0,4,0,1)	4
MagicianLogicBomb,UserAssistSearcher,PrependInfector,YRInfectedPayload(1,4,0,4)	4
ObjLengthLogicBomb,UserAssistSearcher,PrependInfector,YRInfectedPayload(3,4,0,4)	4
Magician2LogicBomb,UserAssistSearcher,PrependInfector,YRInfectedPayload(0,4,0,4)	4
Time1LogicBomb,UserAssistSearcher,PrependInfector,CookiePayload(5,4,0,1)	4
Time2LogicBomb,UserAssistSearcher,PrependInfector,CookiePayload(6,4,0,1)	3
MagicianLogicBomb,UserAssistSearcher,PrependInfector,CheersPayload(1,4,0,0)	3
Magician2LogicBomb,DesktopSearcher,PrependInfector,CheersPayload(0,0,0,0)	3

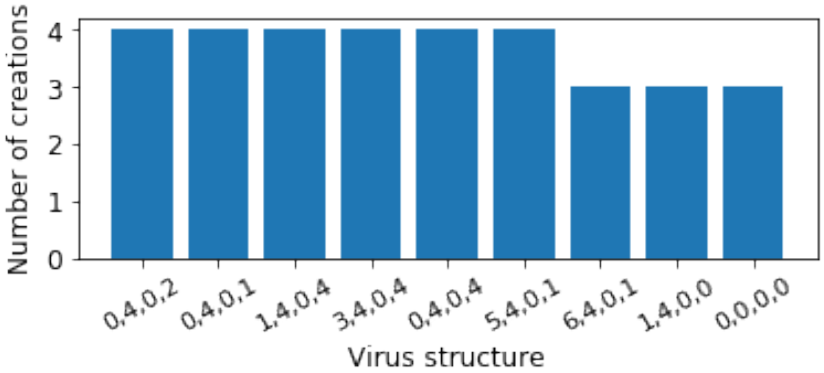

Figure 66: 5-file infection viruses that were created 3 or more times by SOMA

5.3.2.4 Viruses that infects 7 or more files

In Table 34 are shown 5-file infection viruses that infected 7 or more files by randomly selected evolution. The same can be seen in bar plot - Figure 67.

Table 34: 5-file infection viruses that infects 7 or more files by random evolutionary algorithm

Virus structure	**Number of infections**
Magician2LogicBomb,UserAssistSearcher,PrependInfector,YRInfectedPayload(0,4,0,4)	31
Magician2LogicBomb,UserAssistSearcher,PrependInfector,CheersPayload(0,4,0,0)	25
ObjLength2LogicBomb,UserAssistSearcher,PrependInfector,CookiePayload(2,4,0,1)	20
Magician2LogicBomb,UserAssistSearcher,PrependInfector,CookiePayload(0,4,0,1)	18
Magician2LogicBomb,TreeSearcher,PrependInfector,YRInfectedPayload(0,3,0,4)	15
Magician2LogicBomb,TaskBarSearcher,PrependInfector,YRInfectedPayload(0,2,0,4)	15
Magician2LogicBomb,UserAssistSearcher,PrependInfector,GoodXYPayload(0,4,0,2)	12
Time2LogicBomb,UserAssistSearcher,PrependInfector,CookiePayload(6,4,0,1)	10
MagicianLogicBomb,UserAssistSearcher,PrependInfector,CookiePayload(1,4,0,1)	10
MagicianLogicBomb,TreeSearcher,PrependInfector,CheersPayload(1,3,0,0)	10
Magician2LogicBomb,StartMenuSearcher,PrependInfector,GoodXYPayload(0,1,0,2)	10
MagicianLogicBomb,UserAssistSearcher,PrependInfector,IGotYouPayload(1,4,0,3)	10
Magician2LogicBomb,StartMenuSearcher,PrependInfector,CookiePayload(0,1,0,1)	10
Time1LogicBomb,UserAssistSearcher,PrependInfector,CookiePayload(5,4,0,1)	8
RandomLogicBomb,UserAssistSearcher,PrependInfector,GoodXYPayload(4,4,0,2)	8

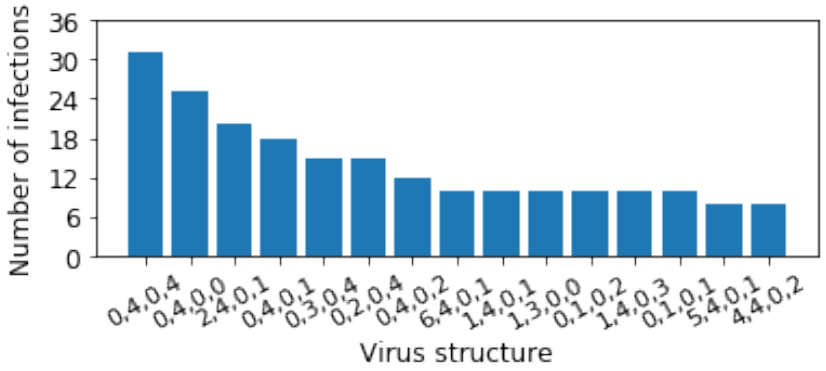

Figure 67: 5-file infection viruses that infects 7 or more files by random evolutionary algorithm

In Table 35 are shown 5-file infection viruses that infected 7 or more files by DE. The same can be seen in bar plot - Figure 68.

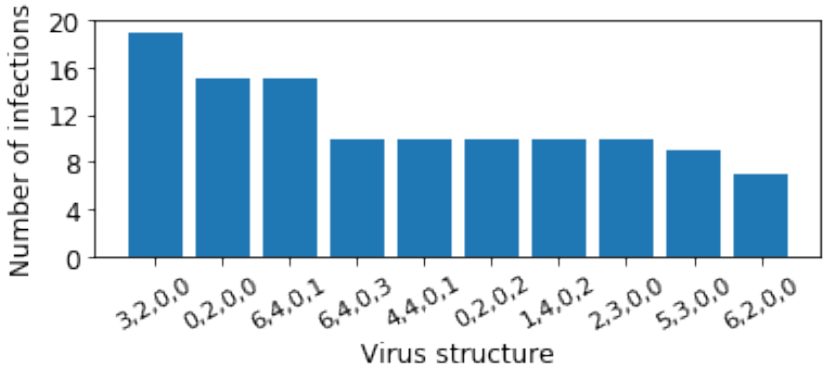

Figure 68: 5-file infection viruses that infects 7 or more files by DE

Table 35: 5-file infection viruses that infects 7 or more files by DE

Virus structure	**Number of infections**
ObjLengthLogicBomb,TaskBarSearcher,PrependInfector,CheersPayload(3,2,0,0)	19
Magician2LogicBomb,TaskBarSearcher,PrependInfector,CheersPayload(0,2,0,0)	15
Time2LogicBomb,UserAssistSearcher,PrependInfector,CookiePayload(6,4,0,1)	15
Time2LogicBomb,UserAssistSearcher,PrependInfector,IGotYouPayload(6,4,0,3)	10
RandomLogicBomb,UserAssistSearcher,PrependInfector,CookiePayload(4,4,0,1)	10
Magician2LogicBomb,TaskBarSearcher,PrependInfector,GoodXYPayload(0,2,0,2)	10
MagicianLogicBomb,UserAssistSearcher,PrependInfector,GoodXYPayload(1,4,0,2)	10
ObjLength2LogicBomb,TreeSearcher,PrependInfector,CheersPayload(2,3,0,0)	10
Time1LogicBomb,TreeSearcher,PrependInfector,CheersPayload(5,3,0,0)	9
Time2LogicBomb,TaskBarSearcher,PrependInfector,CheersPayload(6,2,0,0)	7

In Table 36 are shown 5-file infection viruses that infected 7 or more files by GA. The same can be seen in bar plot - Figure 69.

Table 36: 5-file infection viruses that infects 7 or more files by GA

Virus structure	**Number of infections**
Magician2LogicBomb,UserAssistSearcher,PrependInfector,CookiePayload(0,4,0,1)	30
Magician2LogicBomb,UserAssistSearcher,PrependInfector,CheersPayload(0,4,0,0)	21
Magician2LogicBomb,UserAssistSearcher,PrependInfector,GoodXYPayload(0,4,0,2)	15
Time2LogicBomb,UserAssistSearcher,PrependInfector,CheersPayload(6,4,0,0)	15
ObjLengthLogicBomb,UserAssistSearcher,PrependInfector,CheersPayload(3,4,0,0)	15
Time2LogicBomb,UserAssistSearcher,PrependInfector,YRInfectedPayload(6,4,0,4)	15
ObjLengthLogicBomb,UserAssistSearcher,PrependInfector,GoodXYPayload(3,4,0,2)	14
Magician2LogicBomb,UserAssistSearcher,PrependInfector,IGotYouPayload(0,4,0,3)	10
Time2LogicBomb,UserAssistSearcher,PrependInfector,IGotYouPayload(6,4,0,3)	10
RandomLogicBomb,UserAssistSearcher,PrependInfector,IGotYouPayload(4,4,0,3)	10
Magician2LogicBomb,UserAssistSearcher,PrependInfector,YRInfectedPayload(0,4,0,4)	10
Time1LogicBomb,UserAssistSearcher,PrependInfector,IGotYouPayload(5,4,0,3)	10
MagicianLogicBomb,UserAssistSearcher,PrependInfector,IGotYouPayload(1,4,0,3)	10
MagicianLogicBomb,UserAssistSearcher,PrependInfector,YRInfectedPayload(1,4,0,4)	10
Time1LogicBomb,UserAssistSearcher,PrependInfector,CheersPayload(5,4,0,0)	8

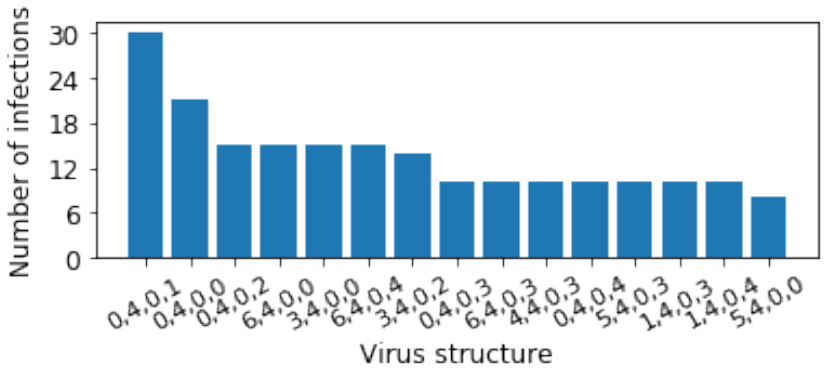

Figure 69: 5-file infection viruses that infects 7 or more files by GA

In Table 37 are shown 5-file infection viruses that infected 7 or more files by PSO. The same can be seen in bar plot - Figure 70.

Table 37: 5-file infection viruses that infects 7 or more files by PSO

Virus structure	**Number of infections**
Magician2LogicBomb,UserAssistSearcher,PrependInfector,YRInfectedPayload(0,4,0,4)	55
Magician2LogicBomb,UserAssistSearcher,PrependInfector,CheersPayload(0,4,0,0)	45
Magician2LogicBomb,UserAssistSearcher,PrependInfector,GoodXYPayload(0,4,0,2)	36
Magician2LogicBomb,UserAssistSearcher,PrependInfector,IGotYouPayload(0,4,0,3)	35
MagicianLogicBomb,UserAssistSearcher,PrependInfector,CookiePayload(1,4,0,1)	20
MagicianLogicBomb,UserAssistSearcher,PrependInfector,YRInfectedPayload(1,4,0,4)	20
Magician2LogicBomb,UserAssistSearcher,PrependInfector,CookiePayload(0,4,0,1)	20
ObjLengthLogicBomb,UserAssistSearcher,PrependInfector,CookiePayload(3,4,0,1)	20
MagicianLogicBomb,UserAssistSearcher,PrependInfector,IGotYouPayload(1,4,0,3)	15
MagicianLogicBomb,UserAssistSearcher,PrependInfector,CheersPayload(1,4,0,0)	10
Magician2LogicBomb,TreeSearcher,PrependInfector,GoodXYPayload(0,3,0,2)	10
ObjLengthLogicBomb,UserAssistSearcher,PrependInfector,GoodXYPayload(3,4,0,2)	10
Time1LogicBomb,UserAssistSearcher,PrependInfector,GoodXYPayload(5,4,0,2)	10
Magician2LogicBomb,TreeSearcher,PrependInfector,CookiePayload(0,3,0,1)	9

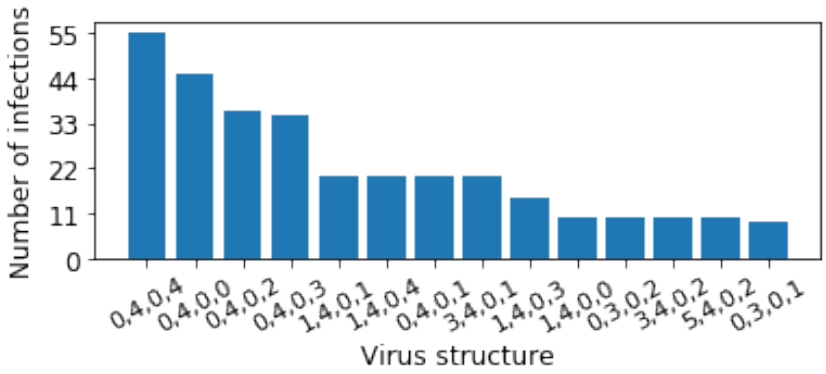

Figure 70: 5-file infection viruses that infects 7 or more files by PSO

In Table 38 are shown 5-file infection viruses that infected 7 or more files by SOMA. The same can be seen in bar plot - Figure 71.

Table 38: 5-file infection viruses that infects 7 or more files by SOMA

Virus structure	Number of infections
Magician2LogicBomb,UserAssistSearcher,PrependInfector,GoodXYPayload(0,4,0,2)	20
Magician2LogicBomb,UserAssistSearcher,PrependInfector,CookiePayload(0,4,0,1)	20
MagicianLogicBomb,UserAssistSearcher,PrependInfector,YRInfectedPayload(1,4,0,4)	20
Magician2LogicBomb,UserAssistSearcher,PrependInfector,YRInfectedPayload(0,4,0,4)	20
ObjLengthLogicBomb,UserAssistSearcher,PrependInfector,YRInfectedPayload(3,4,0,4)	18
Time1LogicBomb,UserAssistSearcher,PrependInfector,CookiePayload(5,4,0,1)	16
Time2LogicBomb,UserAssistSearcher,PrependInfector,CookiePayload(6,4,0,1)	15
MagicianLogicBomb,UserAssistSearcher,PrependInfector,CheersPayload(1,4,0,0)	15
Magician2LogicBomb,DesktopSearcher,PrependInfector,CheersPayload(0,0,0,0)	15
Magician2LogicBomb,StartMenuSearcher,PrependInfector,CheersPayload(0,1,0,0)	10
Magician2LogicBomb,UserAssistSearcher,PrependInfector,CheersPayload(0,4,0,0)	10
ObjLength2LogicBomb,UserAssistSearcher,PrependInfector,GoodXYPayload(2,4,0,2)	10
Magician2LogicBomb,TreeSearcher,PrependInfector,IGotYouPayload(0,3,0,3)	10
Time1LogicBomb,UserAssistSearcher,PrependInfector,GoodXYPayload(5,4,0,2)	10
MagicianLogicBomb,TreeSearcher,PrependInfector,IGotYouPayload(1,3,0,3)	10
RandomLogicBomb,UserAssistSearcher,PrependInfector,CheersPayload(4,4,0,0)	8
Time2LogicBomb,UserAssistSearcher,PrependInfector,IGotYouPayload(6,4,0,3)	7

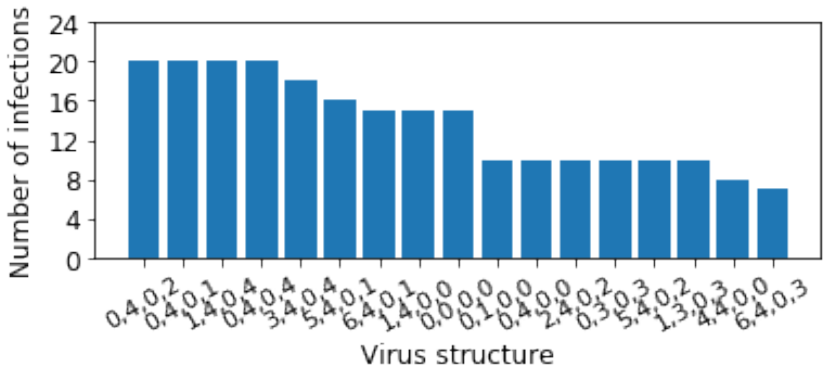

Figure 71: 5-file infection viruses that infects 7 or more files by SOMA

5.3.2.5 Infection visualization

5.3.2.6 Infection visualization

For infection visualization were generated files in Gephi (.gdf) format. All this files are available as attachment in data folder of this work. For visualization of 5-file infect virus was chosen data "data5-random-1.gdf" which means:

- data3 - data from virus that is 5-file infection
- random - evolutionary algorithm is randomly selected
- 1 - it was 1st out of 5 iterations of scenario

For visualization then were created 5 images showing testing files on file system:

- Groups with node names (Figure 72) - shows application group (Program/Game/Work/-Download) and application name
- Groups (Figure 73) - shows application group (Program/Game/Work/Download)
- Spreading heat-map (Figure 74) - shows infection progression
- VictimSearchers (Figure 75) - shows which type of VictimSearcher was used by node (virus) to infect others
- Evolution (Figure 76) - shows which evolution created node (virus)

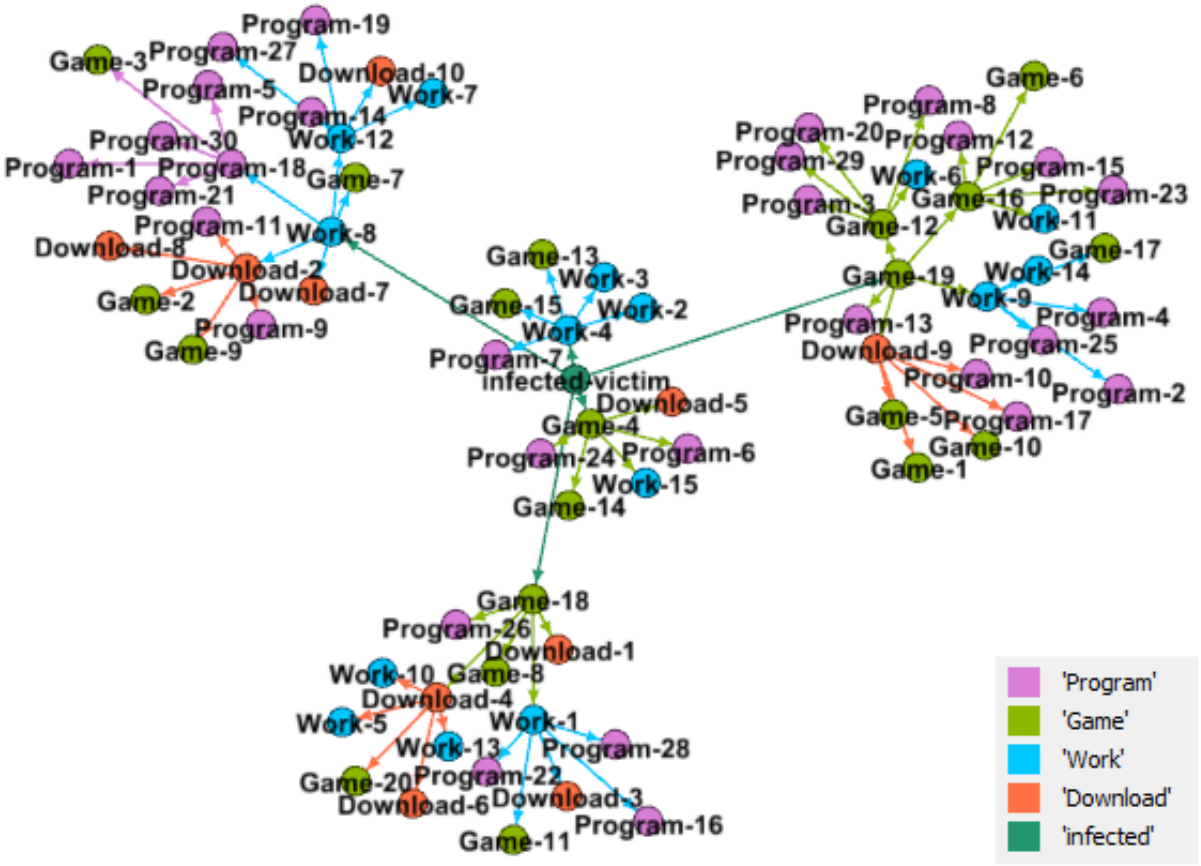

Figure 72: Application group distinguished by color and application names (data5-random-1.gdf)

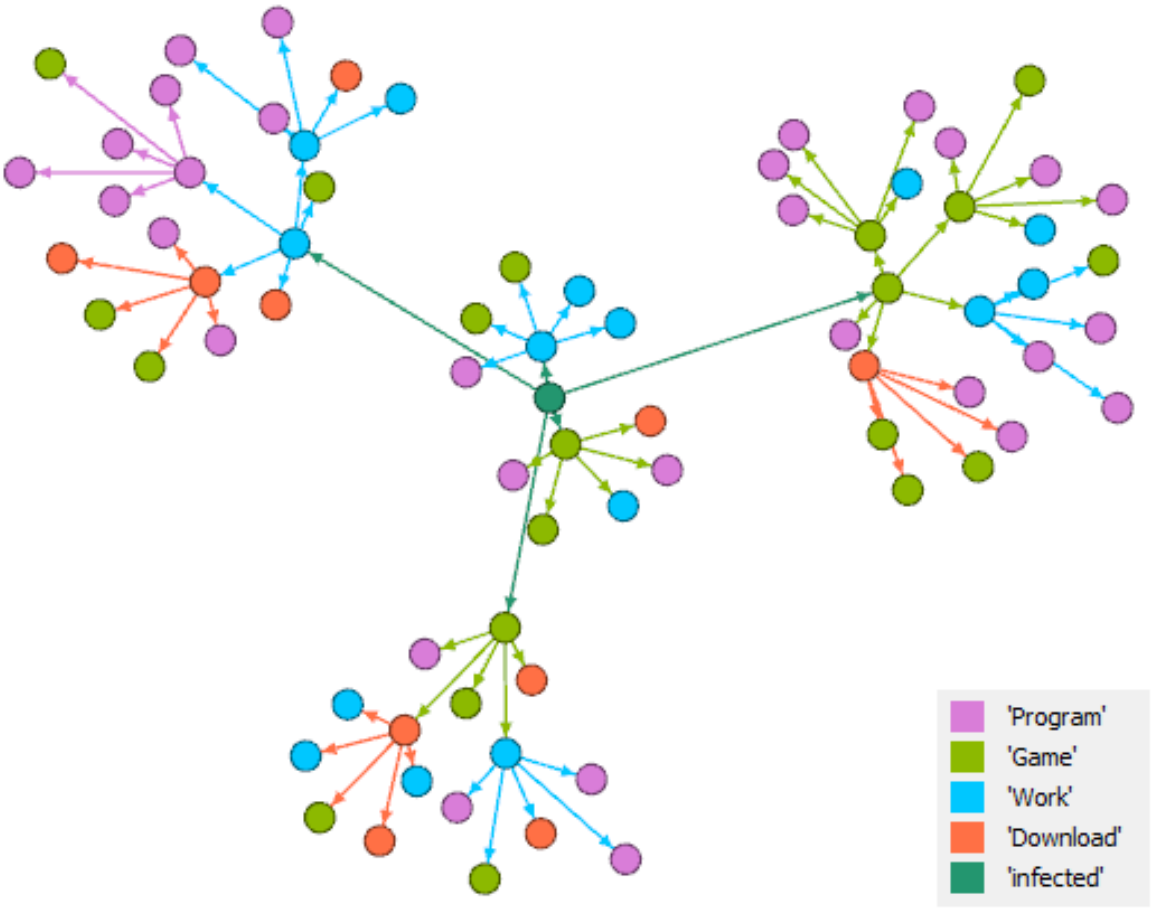

Figure 73: Application group distinguished by color (data5-random-1.gdf)

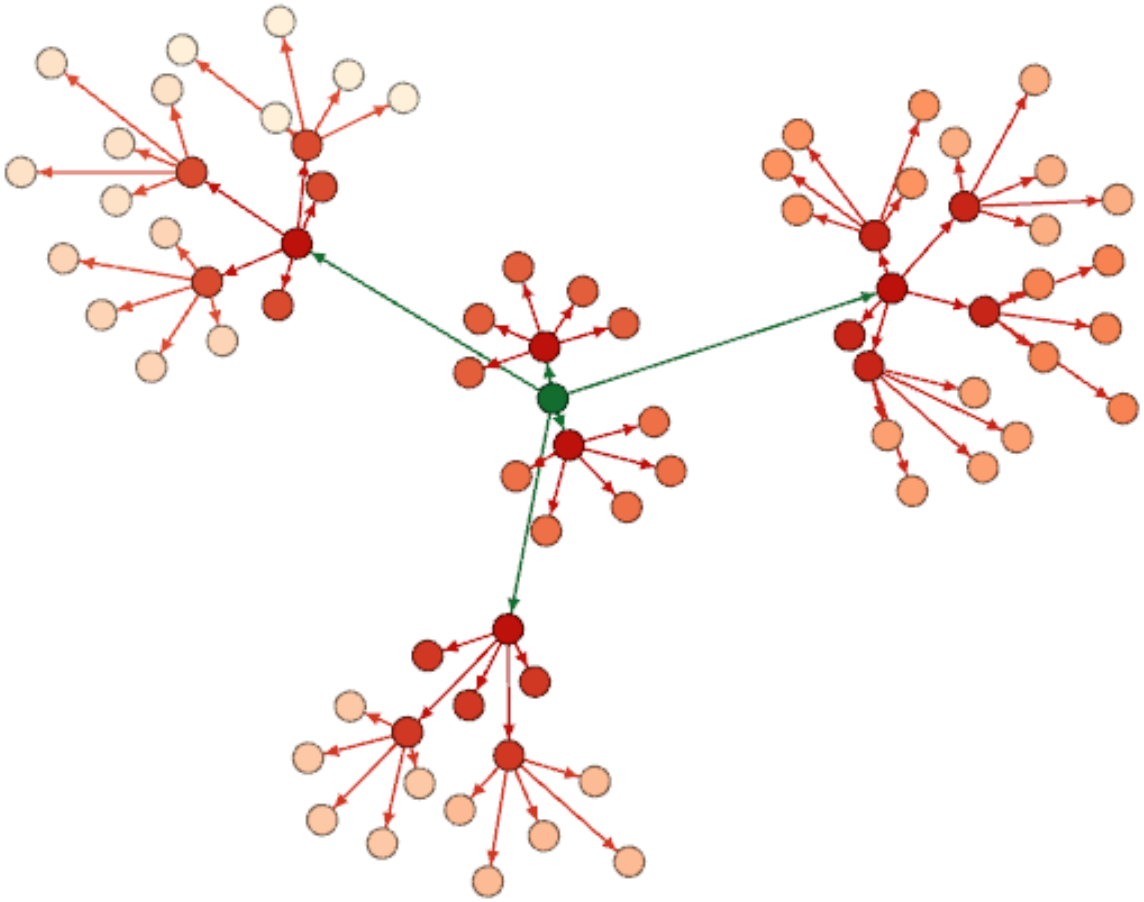

Figure 74: Heat-map of applications infections. Darker nodes were infected earlier. Green node is "infected-victim.exe" first infected file. (data5-random-1.gdf)

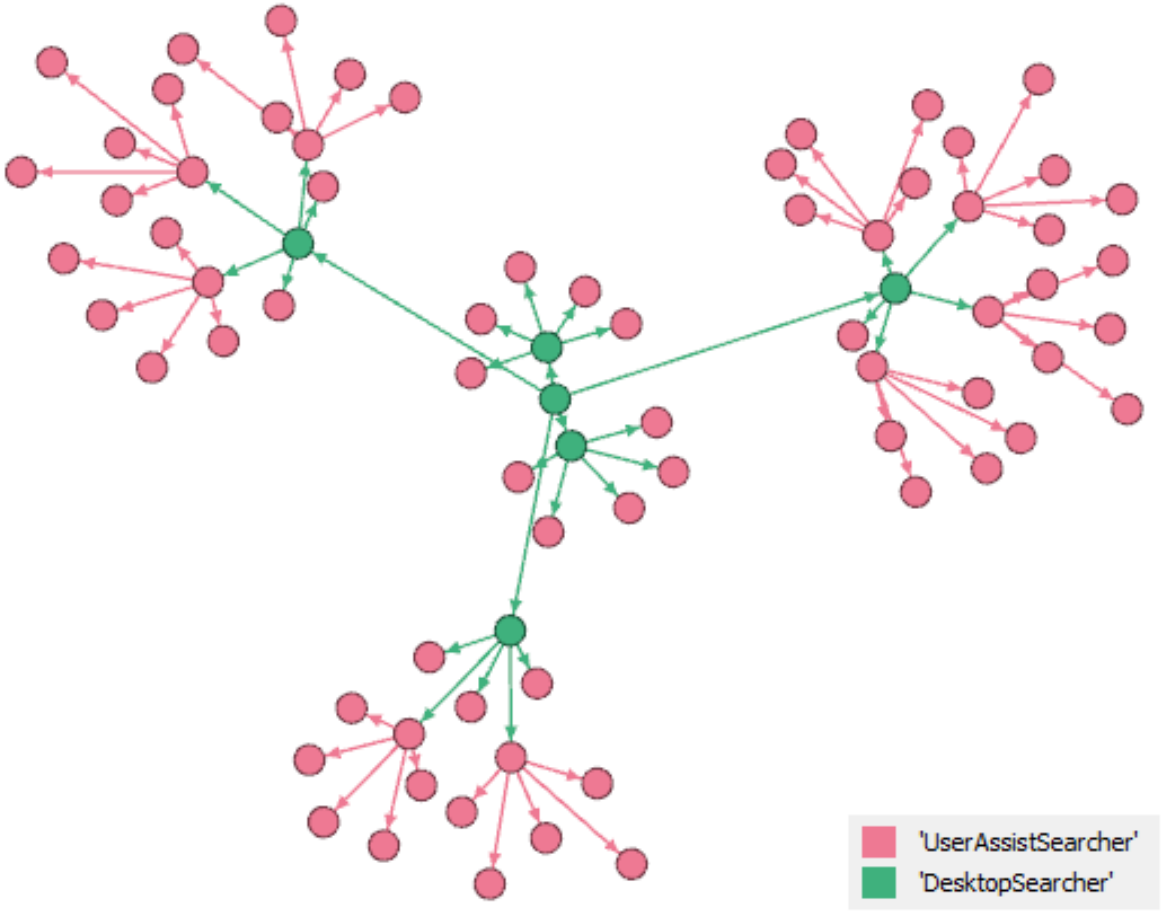

Figure 75: Node (virus) VictimSearcher distinguished by color (data5-random-1.gdf)

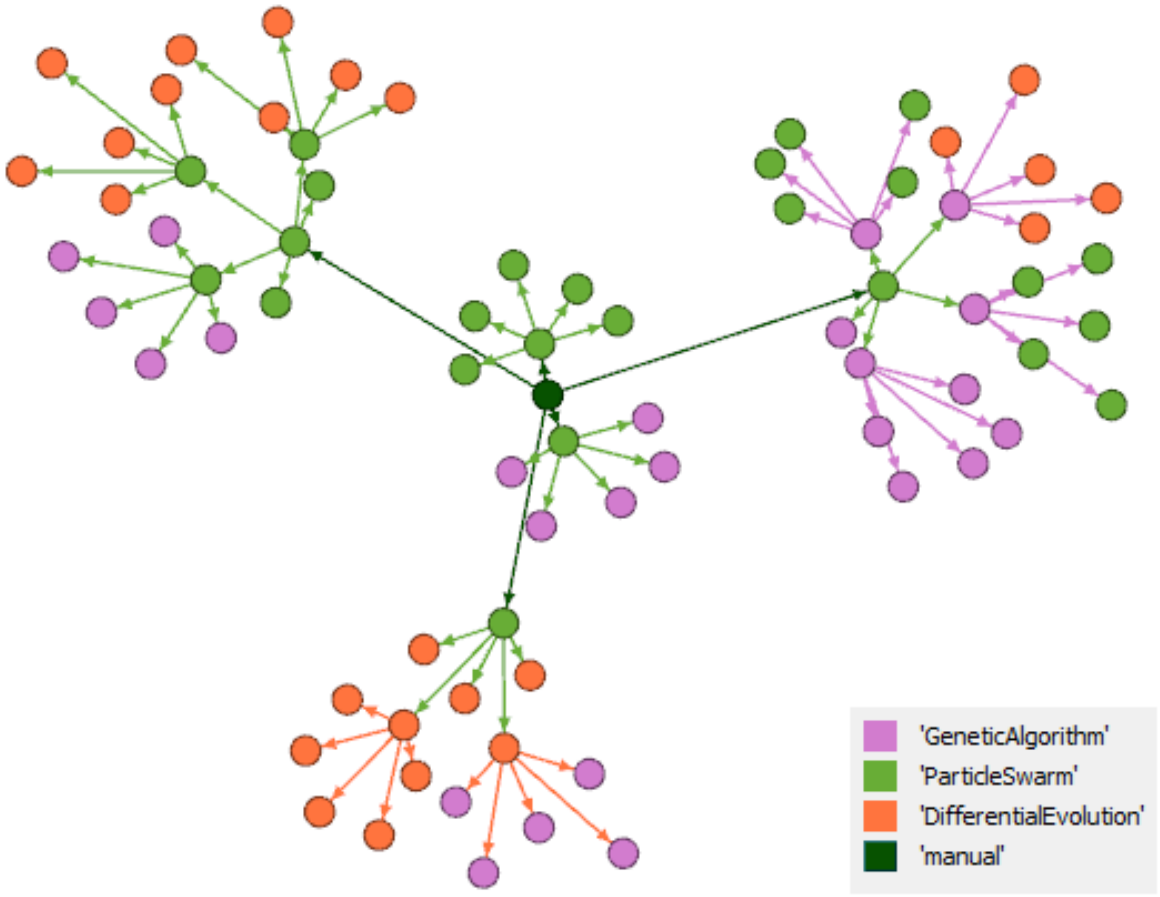

Figure 76: By which evolutionary algorithm was node (virus) created distinguished by color (data5-random-1.gdf)

6 Conclusion

The aim of this master thesis are 3 main things: getting the knowledge how viruses (especially metamorphic) and evolutionary algorithms, make functional evolvable virus and perform experiment with created virus. All these thing were done and are described in this text part of the master thesis.

In the first part of this master thesis are described principles how in general evolutionary algorithms works and what they needed to work. There are also described specific algorithms that were chosen to implementation in this work more in detail and with supplementation of pseudo-code. Next are described various types of malware with description what they do. In the same part are described virus defense techniques with brief description too. The last part of theoretical part is description of performed experiments with using of evolvable malware or evolvable algorithms to create malware or its functionality.

The second part is focused on practical making of evolvable malware. There are description of individual modules of the created virus as well description of techniques that are used by these modules. In this section is also described process of deployment of virus and highlighted things that needs to be done for successful deployment.

The third part is focused on experiment that was done with created virus. There is described the testing environment, the testing scenario and the course of the experiment in depth. Also there is described application that was created to help automate testing process. Next this section contains results of performed experiment in the form of tables, graphs and pictures.

Overall, I consider this work as beneficial to my person. It has expanded my insight into how malware works, what techniques it use and how it spread in controlled virtual environment.

References

[1] SZOR, Peter. Počítačové viry: analýza útoku a obrana. 1. Brno: Zoner Press, 2006. Encyklopedie Zoner Press. ISBN 80-868-1504-8.

[2] NOREEN, Sadia, Shafaq MURTAZA, M. Zubair SHAFIQ and Muddassar FAROOQ. Evolvable malware. Proceedings of the 11th Annual conference on Genetic and evolutionary computation - GECCO '09. New York, New York, USA: ACM Press, 2009, 2009, , 1569-1576. DOI: 10.1145/1569901.1570111. ISBN 9781605583259. Available from: http://portal.acm.org/citation.cfm?doid=1569901.1570111

[3] CANI, Andrea, Marco GAUDESI, Ernesto SANCHEZ, Giovanni SQUILLERO and Alberto TONDA. Towards automated malware creation. Proceedings of the 29th Annual ACM Symposium on Applied Computing - SAC '14 [online]. New York, New York, USA: ACM Press, 2014, 2014, , 157-160. DOI: 10.1145/2554850.2555157. ISBN 9781450324694. Available from: http://dl.acm.org/citation.cfm?doid=2554850.2555157

[4] BABAK BASHARI, Rad, Maslin MASROM and Suhaimi IBRAHIM. Camouflage in malware: from encryption to metamorphism. International Journal of Computer Science and Network Security. 2012, 12(8), 74-83.

[5] POLI, Riccardo, James KENNEDY, Tim BLACKWELL. Particle swarm optimization. Swarm Intelligence. 2007, 1(1), 33-57. DOI: 10.1007/s11721-007-0002-0. ISSN 1935-3812. Available from: http://link.springer.com/10.1007/s11721-007-0002-0

[6] KENNEDY, James. Particle Swarm Optimization. Encyclopedia of Machine Learning. Boston, MA: Springer US, 2011, , 760-766. DOI: 10.1007/978-0-387-30164-8. ISBN 978-0-387-30768-8.

[7] ZELINKA, Ivan, Zuzana OPLATKOVÁ, Miloš ŠEDA, Pavel OŠMERA and František VČELAŘ. Evoluční výpočetní techniky: principy a aplikace. 1. Praha: BEN - technická literatura, 2009. ISBN 978-80-7300-218-3.

[8] NACHENBERG, Carey. Behavior Blocking: The Next Step in Anti-Virus Protection [online]. Symantec Corporation: Symantec Corporation, 2002 [cit. 2019-01-27]. Available from: https://www.symantec.com/connect/articles/behavior-blocking-next-step-anti-virus-protection

[9] History of malicious programs [online]. [cit. 2019-03-27]. Available from: https://encyclopedia.kaspersky.com/knowledge/history-of-malicious-programs/

[10] Year 1988 [online]. [cit. 2019-03-27]. Available from: https://encyclopedia.kaspersky.com/knowledge/year-1988/

[11] KONSTANTINOU, Evgenios. Metamorphic Virus: Analysis and Detection [online]. Royal Holloway, University of London, 2008 [cit. 2019-04-02]. Available from: https://www.csee.umbc.edu/courses/undergraduate/426/fall14/lectures/l07/RHUL-MA-2008-02.pdf. Technical Report. Royal Holloway, University of London. Supervisor Stephen Wolthusen.

A The list of the attachments

1. Visual Studio project of created virus
2. Visual Studio project of FirstVirusInfector application
3. Visual studio project of Tester application
4. virus.zip archive which contains files that are used in the last version of the virus (for showing file hierarchy inside archive)
5. Gathered data from the experiment
6. Microsoft Excel file with metric for computing modules distance (distanceMatrix.xlsx)